

ASP.NET 4.5 网站开发与应用实践教程

- **构思独特**，所有案例来自一线实战场景；
- **实用性强**，将抽象的理论结合到实战案例上；
- **内容全面**，结构清晰，体例丰富。

李振　郭旭辉　编著





ASP.NET 4.5
网站开发与应用实践教程

◎ 李振　郭旭辉　编著

清华大学出版社
北　京

内 容 简 介

本书结合教学特点进行编写，全面讲述 ASP.NET 网站开发技术。全书共分为 17 章，内容包括搭建 ASP.NET 4.5 开发环境、Web 窗体结构和常用页面指令、内置请求和处理对象、使用导航控件和母版页模板、验证控件、ADO.NET 数据库编程、数据绑定、GridView、文件上传与下载以及分页实现等，介绍了 ASP.NET 4.5 的高级开发技术，使用 LINQ to SQL 操作数据库、Ajax 无刷新页面、创建 Silverlight 和 WCF 程序、MVC 4 的简单应用。最后介绍了 ASP.NET 的配置文件以及发布网站的方法。本书示例短小却能体现出知识点，读者能轻松地学习，并灵活地应用到实际的软件项目中去。

本书可作为在校大学生学习使用 ASP.NET 进行课程设计的参考资料，也可以作为高等院校相关专业的教学参考书。

图书在版编目（CIP）数据

ASP.NET 4.5 网站开发与应用实践教程/李振，郭旭辉编著. —北京：清华大学出版社，2017
（清华电脑学堂）
ISBN 978-7-302-42506-9

Ⅰ. ①A…　Ⅱ. ①李…　②郭…　Ⅲ. ①网页制作工具-程序设计-教材　Ⅳ. ①TP393.092

中国版本图书馆 CIP 数据核字（2015）第 316459 号

责任编辑： 夏兆彦　薛　阳
封面设计： 张　阳
责任校对： 徐俊伟
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn, http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 25.5　　**字　　数：** 608 千字
版　　次： 2017 年 2 月第 1 版　　**印　　次：** 2017 年 2 月第 1 次印刷
印　　数： 1～4000
定　　价： 49.00 元

产品编号：060063-01



前　言

FOREWORD

ASP.NET 是目前微软最新的 Web 应用开发平台，ASP.NET 4.5 不仅在语言和技术上弥补了原有 ASP.NET 2.0 的不足，还提供了很多新的控件和特色以提升开发人员的生产力。与之相应，Visual Studio 2012 除了保持与 Visual Studio 旧版本相同的特点之外，也提供了大量新的特色帮助提升开发人员的编程效率。

本书以 Visual Studio 2012 为环境介绍 ASP.NET 4.5 技术的开发与使用，从实用和实际的角度，深入浅出地分析该技术的各个要点。例如，创建 ASP.NET 4.5 应用程序的方法、Web 窗体与网站的区别、内置对象、数据绑定和验证等，同时对新增的 MVC 4、WCF、Silverlight 也有所介绍。

本书内容

全书共分为 17 章，主要内容如下。

第 1 章　搭建 ASP.NET 4.5 的开发环境。本章首先介绍什么是.NET Framework、ASP.NET 的发展历史及 ASP.NET 4.5 的简介，然后介绍如何安装 VS 2012 并创建一个 ASP.NET 应用程序。

第 2 章　ASP.NET Web 窗体页。本章介绍 Web 窗体与网站之间的区别，Web 窗体的运行过程，重点讲解常用的 ASP.NET 页面指令。

第 3 章　Web 服务器控件。本章详细介绍 ASP.NET 中服务器控件的类型和共有属性，常用控件及其应用。

第 4 章　页面请求与响应对象。本章重点介绍 ASP.NET 中与页面请求与响应相关的对象，分别是 Page、Response、Request 和 Server。

第 5 章　数据保存对象。本章重点介绍 ASP.NET 中与保存页面数据有关的对象，分别是 Application、Session、Cookie 和 ViewState。

第 6 章　站点导航控件。本章详细介绍使用站点地图和 XML 文档定义页面逻辑结构，以及站点导航控件的使用。

第 7 章　使用母版页。本章主要介绍母版页的使用、主题的使用、用户控件的创建和使用。

第 8 章　验证用户输入的有效性。本章主要介绍 ASP.NET 中的验证控件，即如何使用验证控件对用户输入的数据进行有效性和安全性验证。

第 9 章　ADO.NET 进行数据库编程。本章首先介绍 ADO.NET 的概念，然后重点对其中的对象进行讲解，包括使用 Connection、Command 和 DataAdapter 等对象进行数据库编程。

第 10 章　数据绑定技术。本章首先简单介绍操作数据时常用的一些绑定方式，然后重点介绍 ASP.NET 提供的数据源控件和数据绑定控件，如 SqlDataSource、Repeater 和

GridView 等。

第 11 章　LINQ 数据处理。本章主要介绍 LINQ 的组成部分、各子句的应用以及 LINQ to SQL 操作数据库的方法。

第 12 章　高级技术应用。本章主要介绍一些 ASP.NET 高级开发技术，如文件上传和下载、文件和目录操作、第三方分页和验证控件等。

第 13 章　Ajax 技术。本章主要介绍 Ajax 技术，包括 Ajax 技术的基础知识、内置的 Ajax 控件和 jQuery 技术的应用等。

第 14 章　Silverlight 入门。本章简单介绍了 Silverlight 的使用，包括 Silverlight 的概念，创建一个 Silverlight 应用程序的过程、XAML 以及与浏览器的交互方法。

第 15 章　ASP.NET MVC 4 框架。本章主要介绍 ASP.NET MVC 4 应用程序的创建方法、MVC 4 应用程序的组成元素及其运行流程。

第 16 章　WCF 入门。本章简单介绍 WCF 的概念、核心元素及 WCF 程序的创建方法，最后介绍防盗链的实现。

第 17 章　配置和部署 ASP.NET 网站。本章主要对 ASP.NET 配置文件 Web.Config 的结构进行详解，同时介绍了发布和复制网站的方法。

本书特色

本书针对初、中级用户量身订做，由浅入深地讲解 ASP.NET 开发动态网站的应用。本书采用大量的范例进行讲解，力求通过实际操作使读者更容易地使用 ASP.NET 开发 Web 应用程序。

❑ **知识点全面**

本书紧紧围绕 ASP.NET 的基础知识开发展开讲解，具有很强的逻辑性和系统性。另外，还将介绍基于 ASP.NET 开发的 LINQ、Ajax、MVC 和 Silverlight 技术。

❑ **实例丰富**

书中各范例和综合实验案例均经过作者精心设计和挑选，它们大多数都是根据作者在实际开发中的经验总结而来，涵盖了在实际开发中所遇到的各种场景。

❑ **应用广泛**

对于精选案例，给出了详细步骤，结构清晰简明，分析深入浅出，而且有些程序能够直接在项目中使用，避免读者进行二次开发。

❑ **基于理论，注重实践**

本书不仅介绍理论知识，而且还介绍过程。在章节的合适位置安排综合应用实例或者小型应用程序，将理论应用到实践当中，来加强读者实际应用能力，巩固开发基础和知识。

❑ **网站技术支持**

读者在学习或者工作的过程中，如果遇到实际问题，可以直接登录 www.ztydata.cn 与我们取得联系，作者会在第一时间内给予帮助。

读者对象

本书可作为在校大学生学习使用 ASP.NET 进行课程设计的参考资料，也适合作为高

等院校相关专业的教学参考书，还可以作为非计算机专业学生学习 ASP.NET 语言的参考书。

除了封面署名人员之外，参与本书编写的人员还有李海庆、王咏梅、康显丽、王黎、汤莉、倪宝童、赵俊昌、方宁、郭晓俊、杨宁宁、王健、连彩霞、丁国庆、牛红惠、石磊、王慧、李卫平、张丽莉、王丹花、王超英、王新伟等。在编写过程中难免会有漏洞，欢迎读者通过清华大学出版社网站 www.tup.tsinghua.edu.cn 与我们联系，帮助我们改正提高。

编　者

目 录

CONTENTS

第 1 章　搭建 ASP.NET 4.5 的开发环境

ASP.NET 是微软公司推出的一项技术，它是一个统一的 Web 开发模型，让开发者使用尽可能少的代码生成企业级 Web 应用程序所必需的各种服务。ASP.NET 作为.NET Framework 的一部分，当开发者编写程序代码时，可以访问.NET Framework 中的类。开发者可以使用与公共语言运行时兼容的任何语言（如 C#和 Visual Basic）编写应用程序的代码。使用这些语言可以开发利用公共语言运行时、类型安全和继承等优点的 ASP.NET 应用程序。

本章简单介绍 ASP.NET 的基础知识，包括.NET Framework、ASP.NET 和 Visual Studio 等内容。

本章学习要点：

- ❑ **掌握.NET Framework 的组成**
- ❑ **了解.NET Framework 4.5 的新增功能**
- ❑ **熟悉 ASP.NET 的发展历史**
- ❑ **了解 ASP.NET 的开发工具**
- ❑ **熟悉 ASP.NET 的特色优势**
- ❑ **熟悉 ASP.NET 4.5 的新增功能**
- ❑ **了解 Visual Studio 的发展历史**
- ❑ **掌握 Visual Studio 2012 的安装**
- ❑ **掌握如何创建 Web 窗体应用程序**

1.1　.NET Framework 概述

.NET Framework 即 Microsoft .NET Framework，它是用于 Windows 的新托管代码编程模型。将.NET Framework 的强大功能与新技术结合起来，可以构建具有视觉上引人注目的用户体验的应用程序，并且能支持各种业务流程。

1.1.1　.NET Framework 的组件

.NET Framework 包含两个组件：公共语言运行时（Common Language Runtime，CLR）和.NET Framework 类库。

1．公共语言运行时

公共语言运行时是.NET Framework 的基础，开发者可以将它看作是一个在执行时管理代码的代理，提供内存管理、线程管理和远程处理等核心服务，并且还强制实施严格

的类型安全以及可提高安全性和可靠性的其他形式的代码准确性。实际上，代码管理的概念是公共语言运行时的基本原则，以公共语言运行时为目标的代码称为托管代码，不以公共语言运行时为目标的代码称为非托管代码。

通过公共语言运行时可以方便地设计出跨语言交互的组件和应用程序，它具有以下几个优点。

（1）使性能得到改进。

（2）能够轻松使用其他语言开发的组件。

（3）类库提供的可扩展类型。

（4）语言功能，如面向对象的编程的继承、接口和重载。

（5）允许创建多线程的可缩放应用程序的显式自由线程处理支持。

（6）结构化异常处理支持。

（7）自定义特性支持。

（8）垃圾回收。

（9）使用委托取代函数指针，从而增强了类型安全和安全性。

2. .NET Framework 类库

.NET Framework 类库是一个综合性的面向对象的可重用类型集合，使用它可以开发多种应用程序，如传统的命令行、图形用户界面应用程序和 Web 窗体应用程序等。

.NET Framework 类库包含一系列丰富的接口、抽象类和非抽象类。System 是.NET Framework 中基本类型的根命名空间，它包括由所有应用程序使用的基本数据类型的类，如 Object、Byte、Array 和 String 等。除了基本类型外，System 命名空间还包含几百个类和多个二级命名空间，如表 1-1 所示为常用的二级命名空间。

表 1-1 System 命名空间下的常用二级命名空间

命名空间	说明
System.Collections	该命名空间包含具有定义各种标准的、专门的和通用的集合对象等功能的类
System.Data	该命名空间包含访问和管理多种不同来源的数据的类
System.Dynamic	该命名空间提供支持动态语言运行时的类和接口
System.Drawing	包含提供与 Windows 图形设备接口的接口类
System.IO	该命名空间包含支持输入和输出的类，包括以同步或异步方式在流中读取和写入数据、压缩流中的数据、创建和使用独立存储区以及处理出入串行端口的数据流等
System.Windows.Forms	定义包含工具箱中的控件及窗体自身的类
System.Net	包含用于网络通信的类或命名空间
System.Linq	该命名空间下的类支持使用语言集成查询（LINQ）的查询
System.Text	System.Text 命名空间包含用于字符编码和字符串操作的类型
System.XML	该命名空间包含用于处理 XML 类型的数据

1.1.2 了解.NET Framework 4.5

从 2002 年发布.NET Framework 1.0 版本开始，多年来，.NET Framework 的版本又

经过多次升级。目前，最新的版本是.NET Framework 4.5.1，它伴随着 Visual Studio 2013 一起发布。但是，由于最新版本的功能和性质并不稳定，因此，本节不介绍.NET Framework 4.5.1 版本，而是对.NET Framework 4.5 进行介绍。

.NET Framework 4.5 伴随着 Visual Studio 2012 一起发布，新增功能如下。

（1）能够在部署期间通过检测并关闭.NET Framework 4 应用程序来减少系统重启。

（2）在 64 位平台下支持大于 2GB 的数组，该功能可在应用程序配置文件中启用。

（3）通过服务器后台垃圾回收提高性能。当开发者在.NET Framework 4.5 中使用服务器垃圾回收时，后台垃圾回收自动启用。

（4）后台实时编译，可在多核处理器上使用此功能改进应用程序性能。

（5）限制正则表达式引擎在超时之前要多久才能尝试解决正则表达式的能力，这需要使用 Regex.MatchTimeout 属性。

（6）定义应用程序域的默认区域性的能力。

（7）Unicode 编码的控制台支持。

（8）支持对区域性字符串排序和比较数据进行版本控制。

（9）Zip 压缩改进，可减少压缩文件的大小。

（10）可以通过 CustomReflectionContext 类自定义用于重写默认反射行为的反射上下文。

（11）支持应用程序的国际域名（IDNA）标准的 2008 版（在 Windows 8 操作系统上使用 System.Globalization.IdnMapping 类时）。

（12）在 Windows 8 中使用.NET Framework 时，将字符串比较委托给操作系统，这将实现 Unicode 6.0。在其他平台上运行时，.NET Framework 包括自己的字符串比较数据，这将实现 Unicode 5.x。

（13）能够在每个应用程序域基础上计算字符串的哈希代码。

（14）类型反射支持 Type 和 TypeInfo 类之间的拆分。

1.2 ASP.NET 概述

ASP.NET 是.NET Framework 的一部分，它是一种使嵌入网页中的脚本可由 Internet 服务器执行的服务器端脚本技术，可以在通过 HTTP 请求文档时再从 Web 服务器上动态创建它们。

1.2.1 发展历史

ASP（Active Server Pages，动态服务器页面）是 ASP.NET 技术的前身。随着 ASP 的发展，它的缺点也日趋显著，这让 ASP 应用程序的维护难度大幅度提高。而且直译式的 VBScript 或 JScript 语言，让效果受到了限制。另外，延展性因为基础架构扩充性不足也受到限制。在这种情况下，急需寻找解决这些缺点的突破口。

1997 年，微软针对 ASP 的缺点准备开发新项目时，刚刚毕业的 ASP.NET 的主要领导人和 IIS 团队合作开发出新一代 ASP 技术的原型，在同年的圣诞节，这项技术被称为

XSP，它的原型就是 Java 语言。

随后，为了将 XSP 移植到 CLR 平台中，XSP 将 XSP 的内核程序全部以 C#语言重新撰写，并更改名称为 ASP+，作为 ASP 技术的后继者，会提供一个简单的移转方法给 ASP 开发者。

然后，ASP+的 Beta 版本在 PDC 2000 中亮相，由 Bill Gates 主讲关键技术的概览，由富士通公司展示使用 COBOL 语言撰写 ASP+应用程序。并且宣布它可以使用 VB.NET、C#、Perl、Nemerle 和 Python 语言开发。

2000 年，微软正式推动.NET 策略，ASP+顺利更改名称为 ASP.NET。经过长时间的开发，ASP.NET 1.0 在 2002 年 1 月 5 日和.NET Framework 1.0 一起亮相。ASP.NET 1.0 正式发布后，其发展速度也异常惊人，2003 年升级为 1.1 版本。

ASP.NET 1.1 版本发布后，更激发了 Web 应用程序开发者对 ASP.NET 的兴趣。由于微软公司提出了"减少 70%代码"的目标，于是 2005 年 11 月微软公司又发布了 ASP.NET 2.0。ASP.NET 2.0 的发布是.NET 技术走向成熟的标志。

伴着强劲的发展势头，2008 年微软推出 ASP.NET 3.5，使网络程序开发更倾向于智能开发。ASP.NET 3.5 是建立在 ASP.NET 2.0 CLR 基础上的一个框架，其底层类库仍调用.NET 2.0 以前封装好的所有类，但在.NET 2.0 的基础上增加多个特性。

2010 年，ASP.NET 4 伴随着.NET Framework 4 在 Visual Studio 2010 中应用。

2012 年，ASP.NET 4.5 伴随着.NET Framework 4.5 在 Visual Studio 2012 中应用。

2013 年，ASP.NET 4.5.1 伴随着.NET Framework 4.5.1 在 Visual Studio 2013 中应用。

提 示

虽然 ASP.NET 4.5.1 **是最新版本，它和**.NET Framework 4.5.1、Visual Studio 2013 **可以在** Windows 8 **系统上应用，但是这还需要一段时间的测试和适应。因此，本书以** ASP.NET 4.5、.NET Framework 4.5 **和** Visual Studio 2012 **为例进行介绍。**

1.2.2 开发工具

ASP.NET 开发的首选语言是 C#以及 VB.NET，同时它也支持多种语言的开发，如 F#、J#、Java、Ruby 和 Delphi 等。

ASP.NET 的网站或者应用程序通常使用微软公司的集成开发环境工具 Visual Studio 进行开发，在开发过程中可以进行编辑。除了 Visual Studio 工具外，还可以使用以下几种工具进行开发。

（1）Dreamweaver：它是集网页制作和网站管理于一身的所见即所得网页编辑器，是第一套针对专业网页设计师特别发展的视觉化网页开发工具，利用它可以轻而易举地制作出跨越平台限制和跨越浏览器限制的充满动感的网页。

（2）SharpDevelop：它是一个用于制作 C#或者 VB.NET 的项目而设计的一个编辑器，同时，这个编辑器本身就是使用 C#开发的，而且公开全部源代码。

（3）MonoDevelop：它是一个跨平台的开放源代码集成开发环境，用来开发 Mono

与.NET Framework 软件。目前支持的语言有 C#、Java、BOO、Nemerle、Visual Basic .NET、CIL、C 与 C++。

（4）WebMatrix：它是微软最新的 Web 开发工具，包含构建网站所需要的一切元素。开发者可以从开源 Web 项目或者内置的 Web 模板开始，也可以直接从无到有编写代码。

（5）Notepad++：它是一款 Windows 环境下免费开源的代码编辑器，支持 C、C++、Java、C#、XML、HTML、PHP 和 JavaScript 语言。

1.2.3 特色优势

ASP.NET 已经成为当前网站开发的技术之一，其特色优势如下。

1. 与浏览器无关

ASP.NET 生成的代码遵循 W3C 标准化组织推荐的 XHTML 的标准，开发者只需要设计一次页面，就可以让该页以完全相同的方式显示、工作在任何浏览器上。

2. 方便设置断点、易于调试

调试一直是程序开发者头痛的一件事，好的调试工具能够使程序达到事半功倍的效果。由于使用的 Web 服务器不受 IDE 的约束，微软有了 IIS 就有了先天的优势，提供了跟踪调试的功能，非常方便代码的找错。

3. 编译后执行，运行效率提高

代码编译是指将代码“翻译”成机器语言，但是在 ASP.NET 中并未直接编译成机器语言，而是先编译成微软中间语言（Microsoft Intermediate Language，MSIL 或 IL），然后由即时编译器（Just In Time，JIT）进一步编译成机器语言。编译好的代码再次运行时不需要重新编译，而是直接使用，这极大地提高了 Web 应用程序的性能。

4. 丰富的控件库

如果要在 JSP 中实现一个树形导航菜单，这需要很多行代码。但是在 ASP.NET 中，程序开发者可以直接使用控件来完成，这样就节省了大量开发时间。内置的控件可以帮助开发者实现许多功能，从而达到减少代码量的目的。

5. 代码后置，使代码更清晰

ASP.NET 采用代码后置技术，将 Web 窗体页面的控件和程序逻辑代码分开显示，这样不仅代码更加清晰，而且有利于开发者阅读和维护。

1.2.4 了解 ASP.NET 4.5

ASP.NET 4.5 是 ASP.NET 开发的一个稳定版本，它支持使用最新的网页开发技术设计现代化的网页。使用网页开发技术设计网页有许多好处，包括更好的安全性、更好的

执行效能、支持开发与部署 Windows Azure 云端网站，而且可以利用最新的 HTML 5 和 CSS 3 网页开发技术设计更好的网页功能。ASP.NET 4.5 的新增功能如下。

（1）支持新的 HTML 5 窗体类型。

（2）在 Web 窗体中支持 Model Binders 模型绑定器。这允许开发者将控件直接绑定到 Model 模型的数据访问方法，并自动对输入的数据做类型转换。

（3）客户端支持独立文件的 JavaScript 验证。

（4）通过打包压缩和精简脚本来改进页性能。

（5）集成 AntiXSS 库，防止 XSS 跨站脚本攻击。

（6）支持 WebSockets 协议。

（7）支持异步读取和写入 HTTP 请求和响应。

（8）支持异步模块和处理程序。

（9）ScriptManager 控件支持内容分布式网络回退。

开发者可以直接使用 ASP.NET 4.5 的技术，也可以将 ASP.NET 1.×~4 网页升级到 ASP.NET 4.5 版本。以 ASP.NET 4 升级到 ASP.NET 4.5 为例，不仅需要在 Visual Studio 2012 中将每个项目的 targetFramework 设置为.NET Framework 4.5 进行翻译，然后在 Web.config 配置文件中设置 compilation 的 targetFramework 为 4.5，还需要在 Web.config 中设置 httpRuntime 的 targetFramework 属性，代码如下。


```
<httpRuntime targetFramework="4.5"/>
```

1.3 Visual Studio 概述

Visual Studio（以下简称 VS）是美国微软公司的开发工具包系列产品，它是一个基本完整的开发工具集，包括整个软件生命周期中所需要的大部分工具，如 UML 工具、代码管理工具和集成开发环境等。

通过 VS 所写的目标代码适用于微软支持的所有平台，包括 Microsoft Windows、Windows Mobile、Windows CE、.NET Framework、.NET Compact Framework 以及 Microsoft Silverlight 和 Windows Phone。

1.3.1 发展历史

1997 年，微软发布了 VS 97，包含面向 Windows 开发使用的 Visual Basic 5.0、Visual C++ 5.0 和面向 Java 开发的 Visual J++以及面向数据开发的 Visual FoxPro。另外，还包含创建 DHTML 所需要的 Visual InterDev，其中 Visual Basic 和 Visual FoxPro 使用单独的开发环境，其他的开发语言使用统一的开发环境。

1998 年，微软发布 VS 6.0，所有开发语言的开发环境版本都升级到 6.0，这也是 Visual Basic 最后一次发布。从下一个版本开始，Microsoft Basic 进化成了一种新的面向对象的语言 Microsoft Basic .NET 2002。

2002 年，随着.NET 口号的提出与 Windows XP/Office XP 的发布，微软发布了 VS.NET

（内部版本号为 7.0）。在这个版本的 VS 中，微软剥离了 Visual FoxPro 作为一个单独的开发环境以 Visual FoxPro 7.0 单独销售，同时取消了 Visual InterDev。与此同时，微软引入了建立在.NET 框架上（版本 1.0）的托管代码机制以及一门新的语言 C#，它是编写.NET 框架的语言。

2003 年，微软对 VS 2002 进行了部分修订，以 VS 2003 的名义发布（内部版本号为 7.1）。Visio 作为使用统一建模语言（Unified Modeling Language，UML）架构应用程序框架的程序被引入，同时被引入的还包括移动设备支持和企业模板。同时，.NET 框架也升级到了 1.1 版本。

2005 年，微软发布了 VS 2005，将.NET 从各种语言的名字中被抹去，但是这个版本的 VS 仍然还是面向.NET 框架的（版本 2.0）。

2007 年，微软发布了 VS 2008。

2010 年，微软发布了 VS 2010 以及.NET Framework 4.0。

2012 年，微软在西雅图发布 VS 2012。

2013 年，微软发布了 VS 2013。

1.3.2 开发版本

最初的 VS 并没有引入.NET Framework 框架，如表 1-2 所示为.NET Framework 引入到 VS 前后时的各个版本。

表 1-2 VS 发展的各个版本

版本名称	内部版本	发布日期	.NET Framework 的支持版本	说明
引入.NET Framework 前				
VS	4.0	1995-04		初版
VS 97	5.0	1997-02		
VS 6.0	6.0	1998-06		
引入.NET Framework 后				
VS .NET 2002	7.0	2002-02-13	1.0	去除 FoxPro 与 J++，以 J#取代 J++
VS .NET 2003	7.1	2003-04-24	1.1	
VS 2005	8.0	2005-11-07	2.0	将.NET 由产品名称中移除
VS 2008	9.0	2007-11-19	2.0、3.0、3.5	去除 J#
VS 2010	10.0	2010-04-12	2.0、3.0、3.5、4.0	加入 F#
VS 2012 RTM	11.0	2012-08-25	2.0、3.0、3.5、4.0、4.5	
VS 2013	12.0	2013-10-17	2.0、3.0、3.5、4.0、4.5、4.5.1	

1.3.3 了解 VS 2012

VS 是目前最流行的 Windows 平台应用程序的集成开发环境。VS 2012 作为一个集成解决方案，适用于无论是个人或者各种规模的开发团队。VS 2012 实现无缝协作，提

高生产效率与专注度，最终将好的主意变成应用。如下列出了 VS 2012 的 6 大特性。

（1）VS 2012 与 VS 2010 相比，最大的特性莫过于对 Windows 8 Metro 开发的支持。

（2）VS 2012 RC 版在界面上比 Beta 版更容易使用，彩色的图标和按照开发、运行和调试等环境区分的颜色方案让人爱不释手。

（3）VS 2012 集成 ASP、.NET MVC 4、全面支持移动和 HTML 5 以及 WF 4.5 等技术。

（4）VS 2012 支持.NET Framework 4.5，与.NET Framework 4.0 相比，.NET Framework 4.5 更多的是完善和改进。

（5）VS 2012 和 TFS 2012 实现更好的生命周期管理。可以说，VS 2012 不仅是一个开发工具，也是团队的管理信息系统。

（6）VS 2012 对系统资源的消耗并不大，但是需要 Windows 7 或 Windows 8 的支持。

1.4 安装 VS 2012

VS 2012 支持.NET Framework 4.5，本节介绍 VS 2012 的具体安装过程。但是在安装之前，首先要了解前期的准备工作。

1.4.1 准备工作

与之前的版本相比，VS 2012 的安装界面和安装过程会有很大的变化，它更加简单化和智能化。在安装 VS 2012 之前，需要在微软的官网下载 VS 2012，如图 1-1 所示。

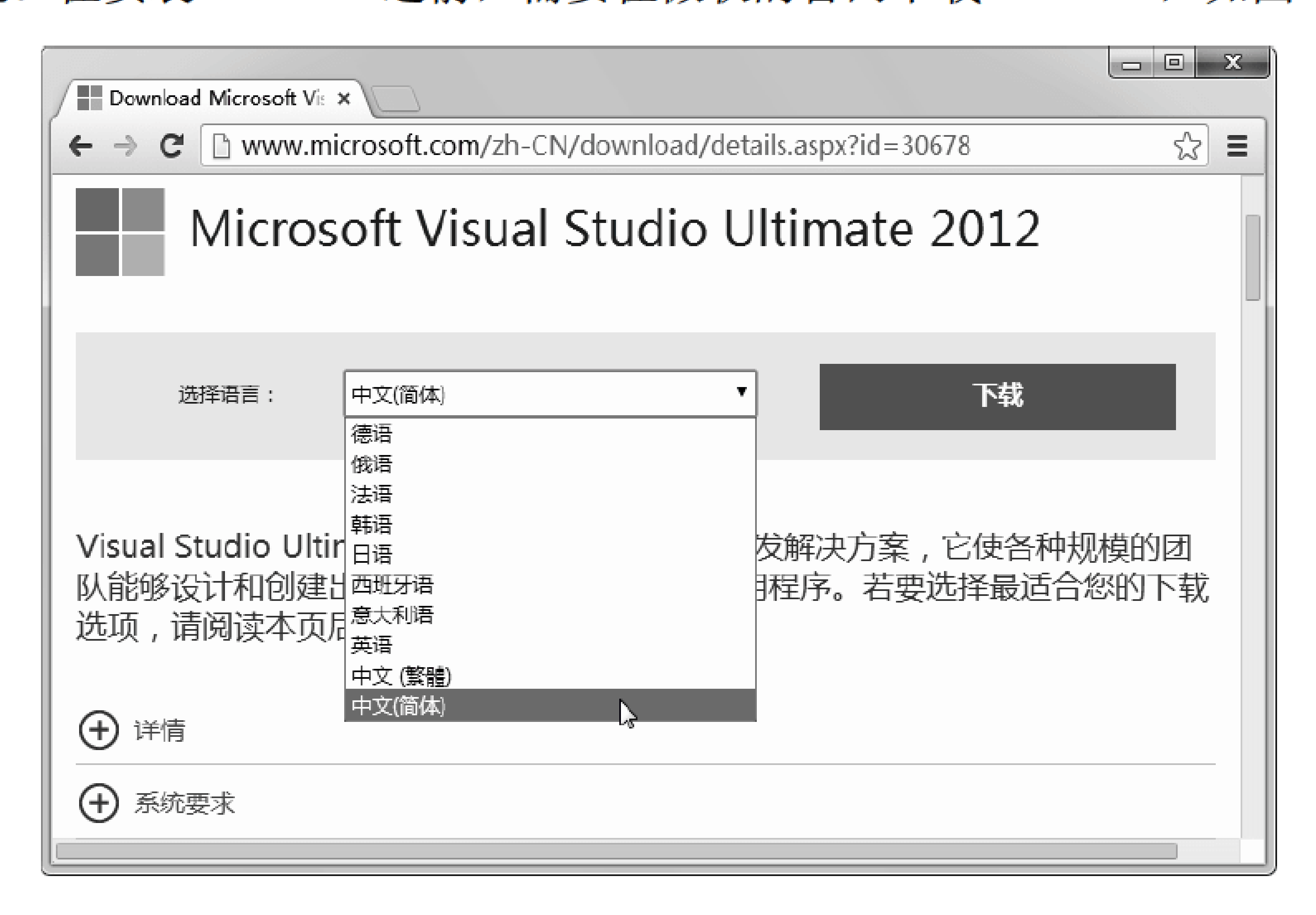


图 1-1 界面下载

在如图 1-1 所示界面中，开发者可以根据需要选择语言，这里选择简体中文版，选择完毕后单击【下载】按钮。如果有需要，开发者可以单击页面中的选项查看详情和系统要求，如图 1-2 所示为安装 VS 2012 时的系统要求。

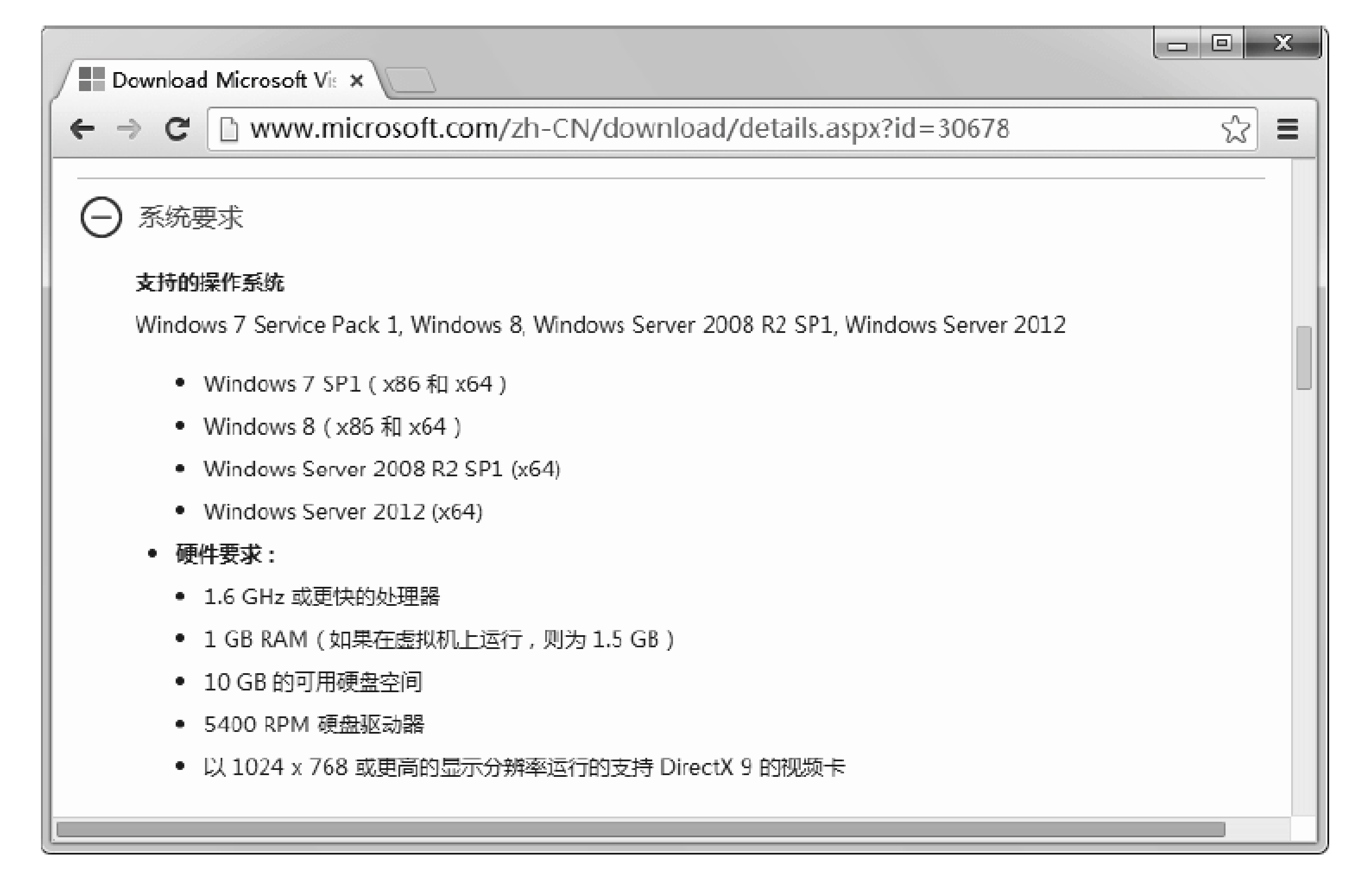


图 1-2　系统要求

1.4.2　安装步骤

在下载 VS 2012 的磁盘目录中打开 VS2012_ULT_chs.iso 文件并解压缩，步骤如下。

（1）打开解压后的文件夹，找到 vs_ultimate.exe 文件并双击，稍等片刻后出现如图 1-3 所示的界面。

（2）在如图 1-3 所示界面中，可以更改 VS 2012 的安装路径，选中第一个复选框表示同意许可条款，如图 1-4 所示。



图 1-3　初始界面



图 1-4　同意条款

（3）单击图 1-4 中的【下一步】按钮，弹出如图 1-5 所示的界面，在该界面中选择要安装的功能。如果不安装某个功能，可以取消勾选该功能前面的复选框。

（4）单击【安装】按钮进行安装，如图 1-6 所示。



图 1-5　选择功能

图 1-6　安装效果

（5）VS 2012 组件安装完毕后的效果如图 1-7 所示。

（6）单击如图 1-7 所示界面中的【启动】按钮，首先进入的界面效果如图 1-8 所示。



图 1-7　安装成功



图 1-8　单击启动

（7）首次运行 VS 2012 时弹出如图 1-9 所示的界面。在如图 1-9 所示的界面中，要求开发者选择默认环境设置，这里选择 Visual C#开发设计，如果不进行设置，直接单击【退出 Visual Studio】按钮。



图 1-9　选择默认环境设置

1.4.3　认识界面

开发者可以单击【开始】|【所有程序】|Microsoft Visual Studio 2012 命令启动 VS 2012，也可以单击桌面上的快捷方式（如果有）直接启动。如图 1-10 所示为 VS 2012 启动成功后的界面。



图 1-10　VS 2012 开发界面

1．菜单栏

菜单栏位于 VS 2012 开发界面的最顶端，它提供一系列默认的工具和可执行的操作。菜单栏中包含多个菜单项，如“文件”“编辑”“视图”“调试”“工具”“帮助”等。

（1）“文件”菜单：包含项目的打开、保存和导出等，和普通软件的“文件”菜单没有区别。

（2）“编辑”菜单：包含常用的查找、替换、删除和格式化等操作。

（3）“视图”菜单：视图就是从整体上对开发界面进行布局，包括一些常用的提示窗口。这个菜单非常重要，如果显示一些错误窗口和资源管理窗口，那么开发者可以直观地了解程序的错误，以及程序所包括的所有文件。常用的视图窗口有“服务器资源管理器”“解决方案管理器”“工具条”“属性窗口”等。

（4）“调试”菜单：开发者在编写代码时用于执行、调试和判断代码，还可以在代码中设置断点，以查看变量的结果。这是开发者经常使用并且必须了解的菜单。

（5）SQL 菜单：它是对项目中当前的数据源进行管理，这些数据源包括数据库、各种服务和对象等。

（6）“工具”菜单：提供 VS 2012 可以支持的所有工具，如果要用菜单中没有的工具，还可以自行添加。

（7）“测试”菜单：开发者可以使用它对项目和类库进行各种测试，及时检查代码错误。

（8）“窗口”菜单：提供一些窗口的布局操作，如浮动、隐藏和拆分等。

（9）“帮助”菜单：提供了前面安装的 MSDN 说明文档的一些操作。

2．工具栏

工具栏一般位于主框架窗口的上部，菜单栏的下方，由一些带图片的按钮组成，当用户用鼠标单击工具栏上的某个按钮时，程序会执行相应的操作。VS 2012 中提供了数十种工具栏，常用的有以下 4 种。

（1）标准工具栏：和其他软件的标准工具栏一样，提供常用的“保存”“打开”“新建”按钮等。

（2）布局工具栏：用来对窗体中的各个设计组件进行统一布局，例如左对齐和居中等。

（3）调试工具栏：实现对代码的执行、中断和逐行执行等功能。当鼠标指针移向某按钮时，还会提示这个按钮的快捷键。

（4）文本编辑器工具栏：在打开窗体设计视图时，该工具栏处于不可用状态。因为它只支持代码文本的编辑，包括代码的缩进、注释和标记等。

3．解决方案资源管理器

解决方案资源管理器类似于 Windows 操作系统的资源管理器，可以在此窗口下查看当前项目所包含的任何资源，例如文件夹、类文件和数据文件等。

解决方案资源管理器在系统中被保存为一个完整的文档，默认扩展名是.sln。在该解决方案管理器下可以包含多种项目，既可以包含 Windows 项目，也可以包含 Web 项目，还可以在 Web 项目中引用 Windows 项目。

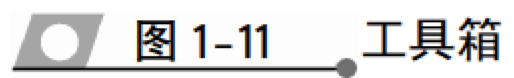
4．工具箱

工具箱中包含 VS 2012 提供的常用控件，如常用的标准控件（如 TextBox 和 Label）、与数据有关的控件（如 GridView）、验证控件（如 RequiredFieldValidator）、导航控件、登录控件、WebParts 控件以及 AJAX 扩展和报表控件等，如图 1-11 所示。

5．【属性】窗口

【属性】窗口用来显示项目、窗体、控件和数据源等所有可视资源的属性。如果要查看某个按钮的名字和字体，可以通过打开【属性】窗口来设置。按 F4 键就可以打开【属性】窗口，或者选中该控件后右击，然后单击【属性】选项查看，如图 1-12 所示为 GridView 控件的【属性】窗口。单击图 1-12 中类似于闪电的符号可以查看 GridView 控件的事件，如图 1-13 所示。




图 1-11　工具箱

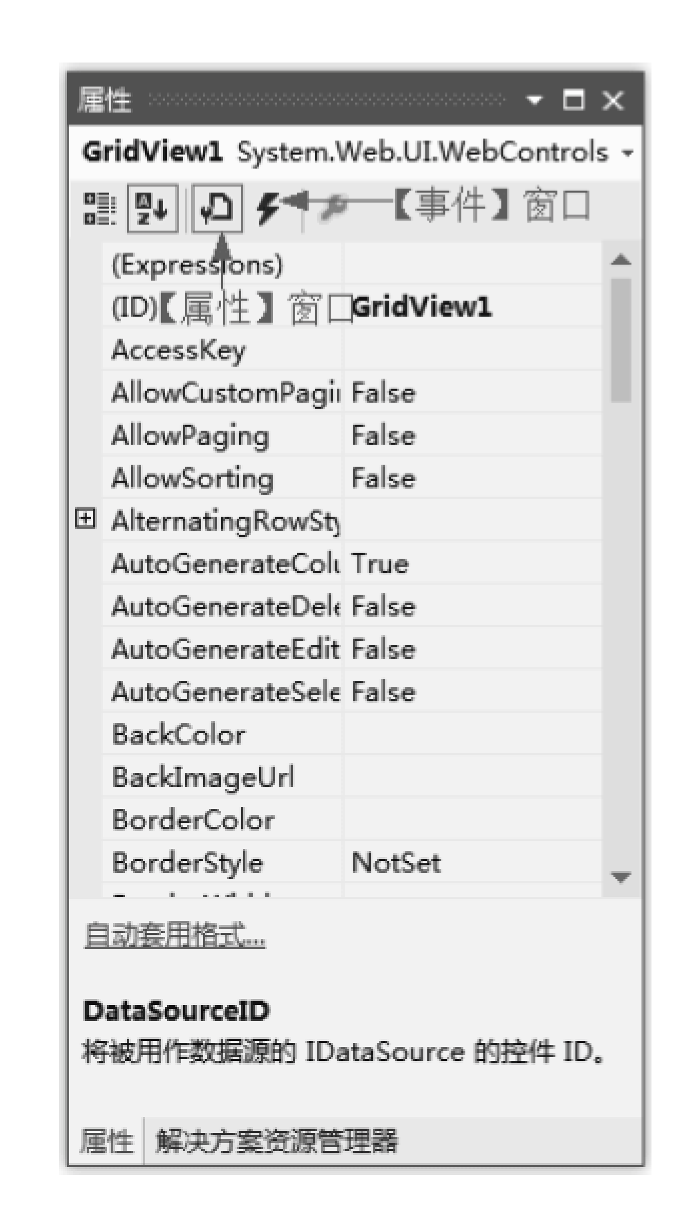


图 1-12　【属性】窗口

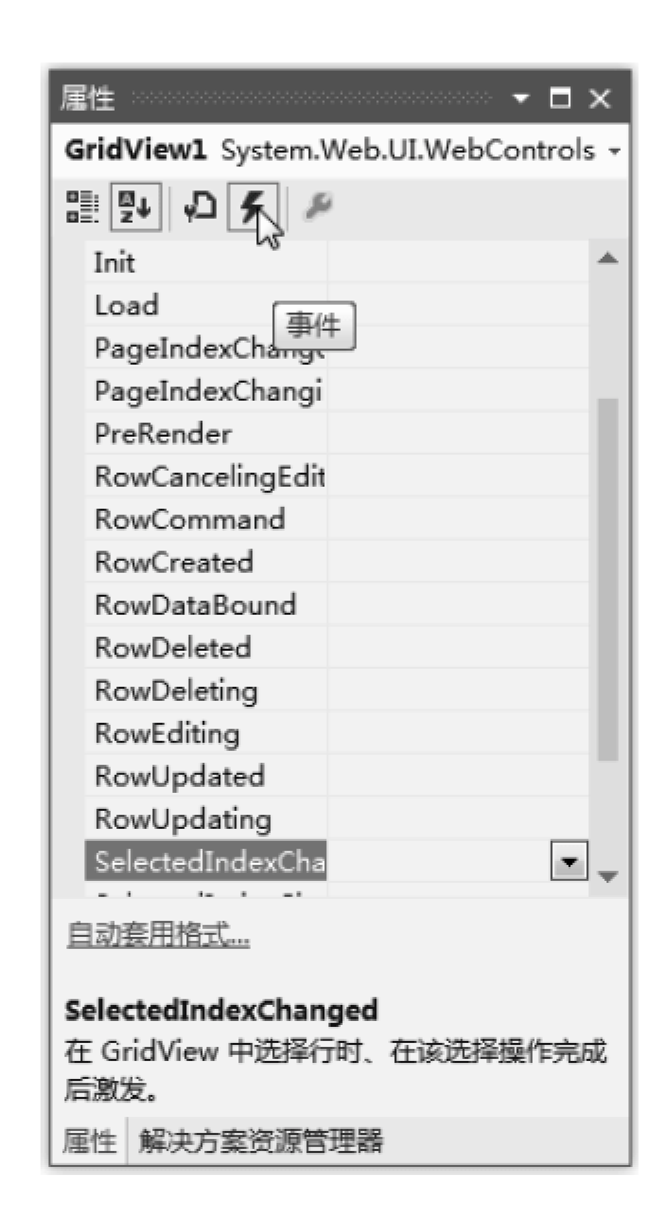


图 1-13　事件列表

6．服务器资源管理器

服务器资源管理器在以前并不常用，但是在 VS 2012 中，它的功能被彻底地挖掘出来。因为 VS 2012 提供了 LINQ to SQL 类，该类必须依靠数据源才可以生成数据库表的映射类，而数据源的管理就在服务器资源管理器中。开发者选择【视图】|【服务器资源管理器】命令或直接按 Ctrl+W+L 键，即可打开服务器资源管理器，如图 1-14 所示。选择【服务器】或【数据连接】右击，可以添加新的服务器或数据连接，如图 1-15 所示。

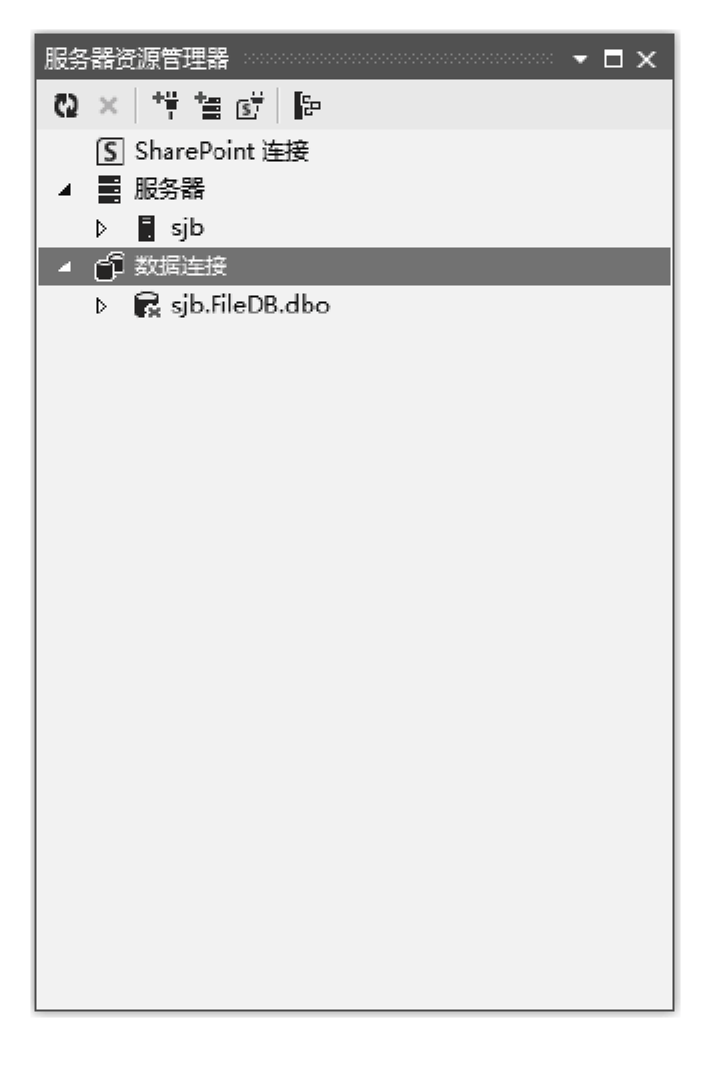


图 1-14　服务器资源管理器

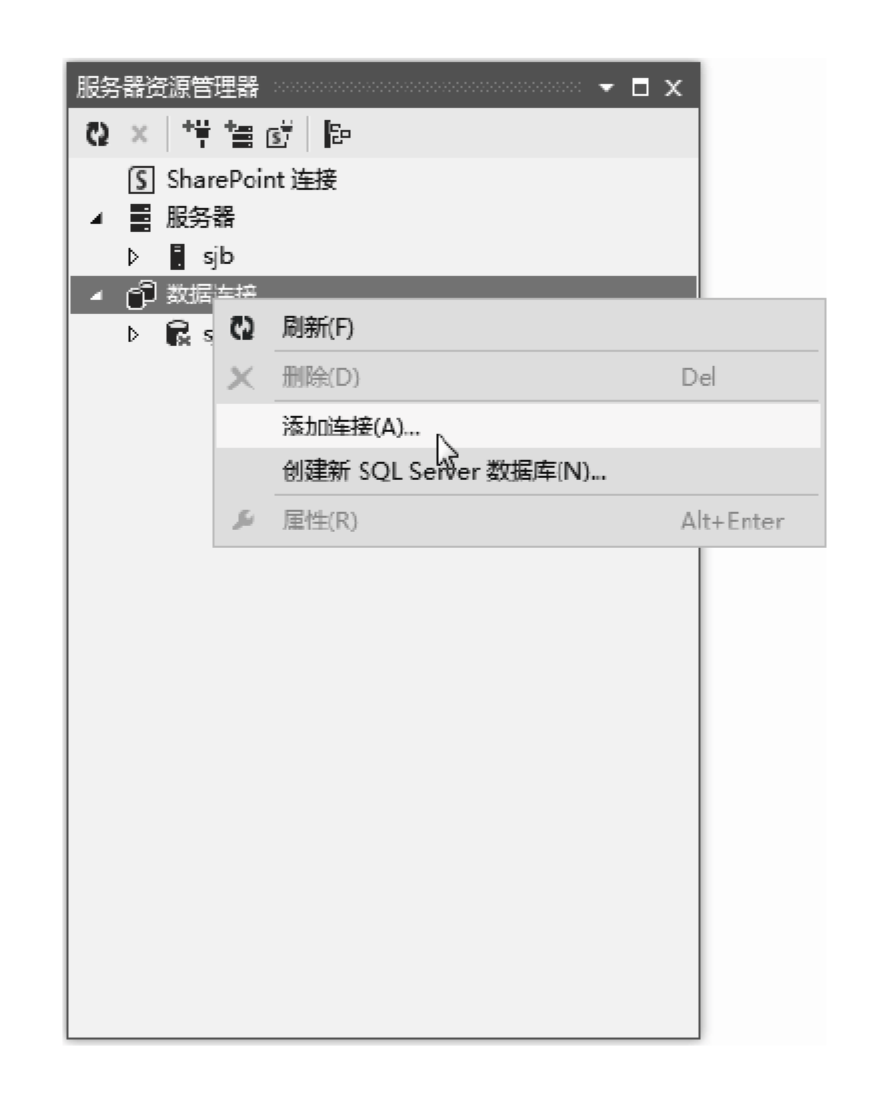


图 1-15　在资源管理器添加连接

1.5　实验指导——创建 ASP.NET Web 窗体应用程序

安装 VS 2012 的目的是为了使用，本节实验指导利用 VS 2012 创建 ASP.NET Web 窗体应用程序。步骤如下。

（1）单击【文件】|【新建】|【项目】命令，弹出如图 1-16 所示的对话框。在该对话框中选择【其他项目类型】|【Visual Studio 解决方案】，输入解决方案的名字和位置，单击【确定】按钮创建一个解决方案。

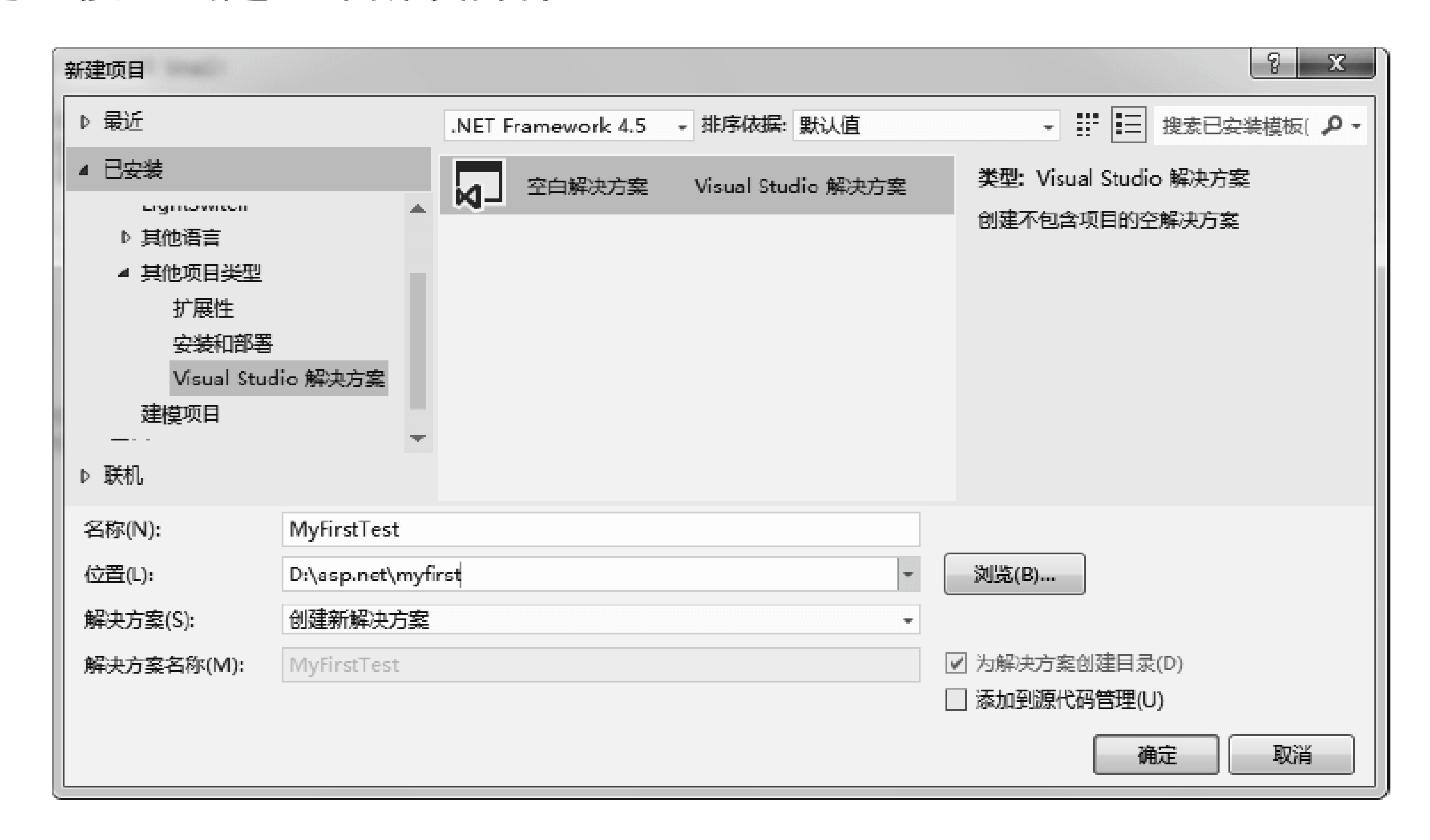


图 1-16　创建解决方案

（2）选中创建的解决方案名称后右击，在弹出的快捷菜单中选择【添加】|【新建项目】命令，弹出【添加新项目】对话框。

（3）在弹出的对话框中找到 Visual C#下的 Web 选项，创建 ASP.NET Web 窗体应用程序，如图 1-17 所示。

图 1-17　创建 ASP.NET Web 窗体应用程序

（4）在图 1-17 中输入应用程序的名称，并选择应用程序的位置，然后单击【确定】按钮，如图 1-18 所示。

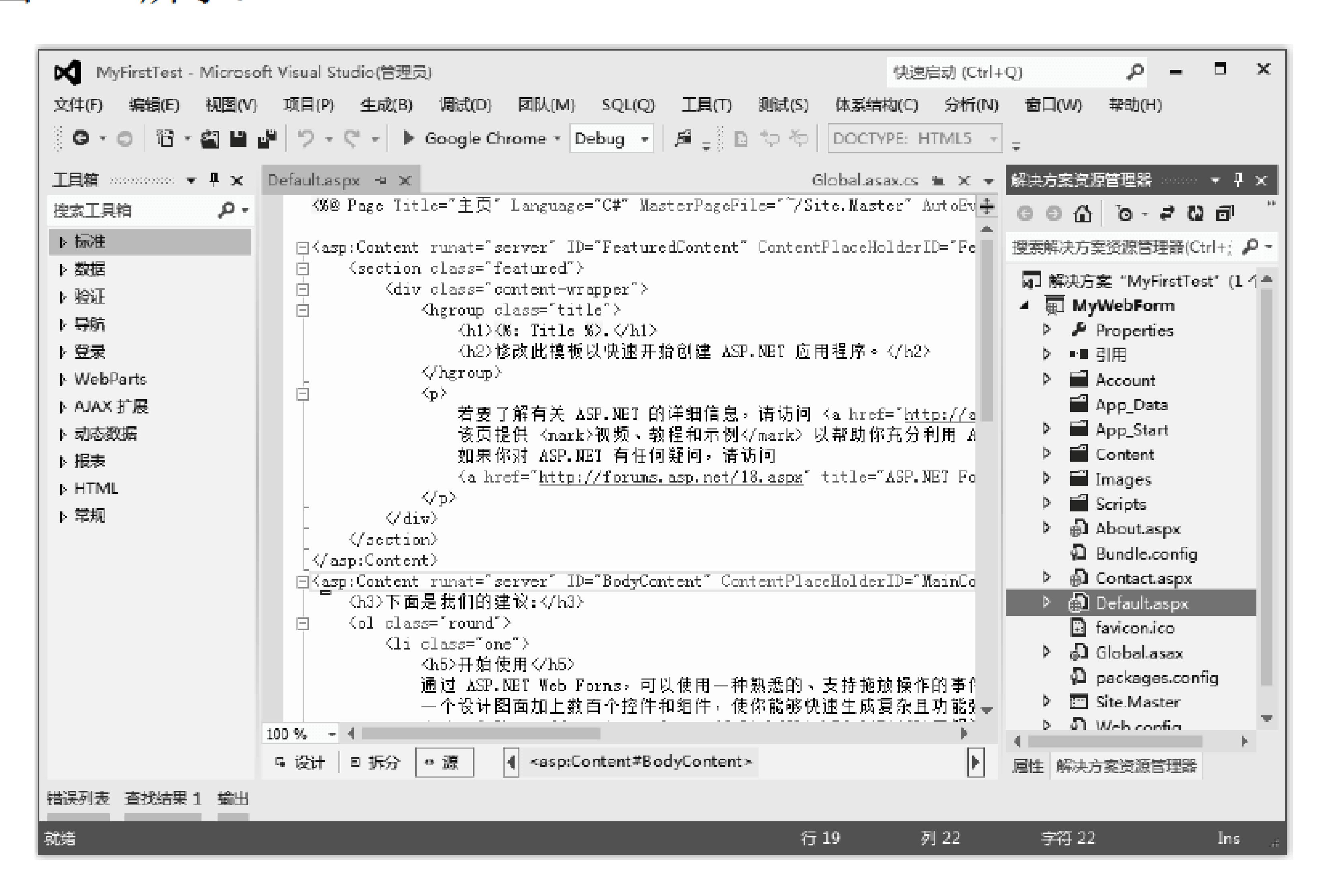


图 1-18　创建程序成功后的界面

（5）创建 ASP.NET Web 窗体应用程序会自动生成多个目录和文件。开发者可以直接运行 Default.aspx 页面查看效果，如图 1-19 所示。



图 1-19　查看 Default.aspx 页面

（6）选中当前应用程序后右击，在快捷菜单中选择【添加】|【添加新建】命令，弹出如图 1-20 所示的对话框。选择 Web 窗体后输入名称“FirstTest”，单击【确定】按钮即可添加。开发者也可以直接选择右键菜单中的【添加】|【Web 窗体】命令添加窗体。

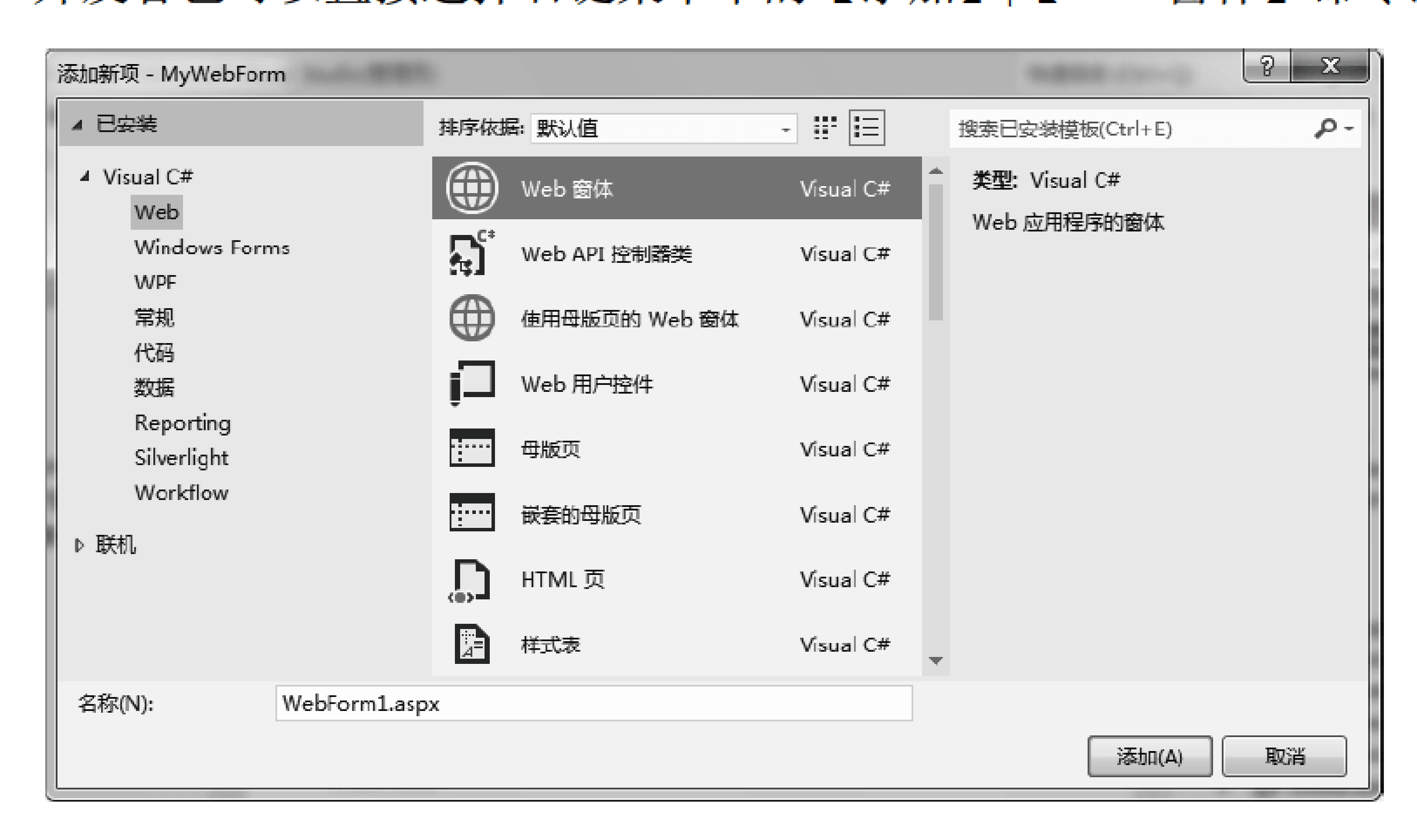


图 1-20　添加 **Web** 窗体

（7）打开 FirstTest.aspx 页面，从【工具箱】中分别拖动 TextBox、Button 和 Label 控件到页面上。

（8）双击页面中的 Button 控件添加 Click 事件，Click 事件代码如下。

```
protected void Button1 Click(object sender, EventArgs e) {
    Label1.Text = "输入的值是：" + TextBox1.Text;
}
```

（9）运行 First.aspx 页面输入内容进行测试，如图 1-21 所示。

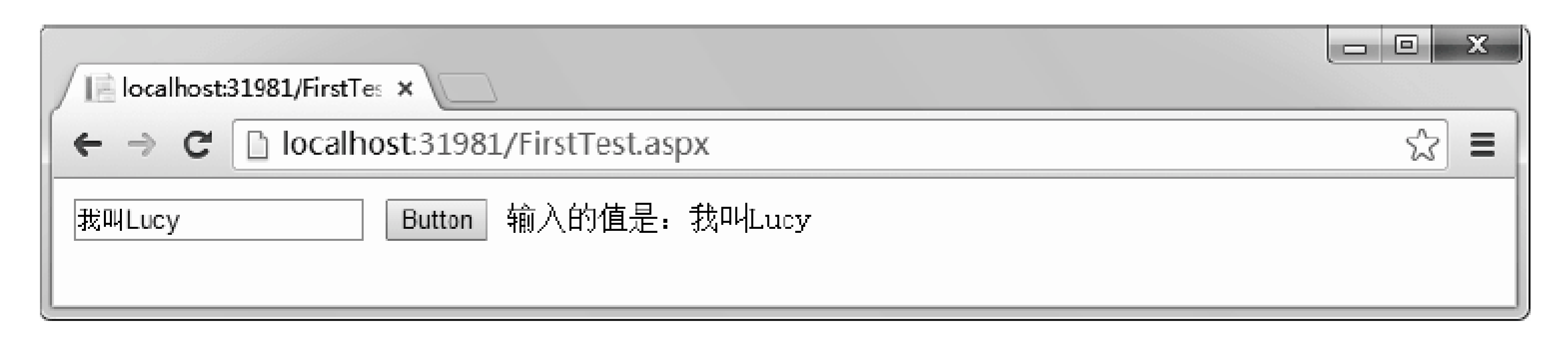


图 1-21　输入内容测试页面

思考与练习

一、填空题

1．__________是.NET Framework 的基础。

2．.NET Framework 中基本类型的根命名空间是__________。

3．VS 是英文__________的简写。

二、选择题

1．关于 ASP.NET 的概述，下面说法正确的是__________。

A．ASP.NET 页面运行时采用先执行后编译的方式，这极大地提高了 Web

应用程序的性能

B．ASP.NET 与 ASP 一样，都不支持调试的功能，因此不能够在页面中设置断点

C．ASP.NET 采用代码后置将 Web 页面元素和程序逻辑分开显示，使代码更加清晰，有利于阅读和维护

D．虽然 ASP.NET 的功能很强大，但是它与浏览器有关

2．下面哪个版本的 VS 没有引入.NET Framework？ __________

A．VS 97

B．VS 2013

C．VS 2012

D．VS 2008

3．VS 2012 的菜单栏中，__________菜单包含常用的查找、替换、删除和格式化等操作。

A．文件

B．编辑

C．工具

D．帮助

4．解决方案资源管理器的默认扩展名是__________。

A．.ashx

B．.slns

C．.aspx

D．.sln

三、简答题

1．简单描述 ASP.NET 的发展历史。

2．请说出 ASP.NET 4.5 的新增功能（至少三点）。

3．概述 VS 2012 的安装步骤。

第 2 章　ASP.NET Web 窗体页

在 ASP.NET 中，发送到客户端浏览器中的网页由.NET Framework 的基类动态生成，该基类是 Web 页面框架中的 Page 类，而一个实例化的 Page 类就是一个 Web 窗体。也就是说，一个 ASP.NET 页面就是一个 Web 窗体。本章简单介绍 ASP.NET Web 窗体的结构，包括页面运行机制和常用指令等。在介绍 Web 窗体页之前，分别创建 Web 窗体应用程序和网站比较它们之间的异同点。

本章学习要点：

- ❑ 掌握 Web 应用程序的创建
- ❑ 掌握 Web 网站的创建
- ❑ 熟悉 Web 应用程序与网站的异同点
- ❑ 了解 Web 窗体页的特点
- ❑ 熟悉 Web 窗体页的元素
- ❑ 了解 Web 窗体页的运行过程
- ❑ 掌握@Page 和@Control 指令
- ❑ 掌握@Register 和@Master 指令
- ❑ 了解 ASP.NET 的其他页面指令

2.1 Web 应用程序和网站

C/S 和 B/S 是应用程序的两种模式，C/S 是客户/服务器程序，而 B/S 是浏览器/服务器应用程序，这类应用程序一般借助于 IE、Chrome 和 Firefox 等浏览器来运行。Web 应用程序一般是 B/S 模式，它是基于 Web 的，而不是采用传统方法运行的。简单来说，Web 应用程序是典型的浏览器/服务器架构的产物。

2.1.1　新建 Web 应用程序

在第 1 章中已经介绍过如何创建一个基于窗体的 Web 应用程序，基于窗体的 Web 应用程序创建完毕后会自动生成一些目录和文件，如图 2-1 所示为基于窗体的 Web 应用程序。

在如图 2-1 所示界面中，包含多个目录和文件，常用的目录和文件说明如下。

（1）Properties 目录：该目录包含一个 AssemblyInfo.cs 文件，这是一个包含程序版本、信息和版权的属性文件。

（2）Account 目录：该目录包含多个 Web 窗体，这是基于窗体创建 Web 应用程序时生成的一个目录，包含用户登录和注册等页面。

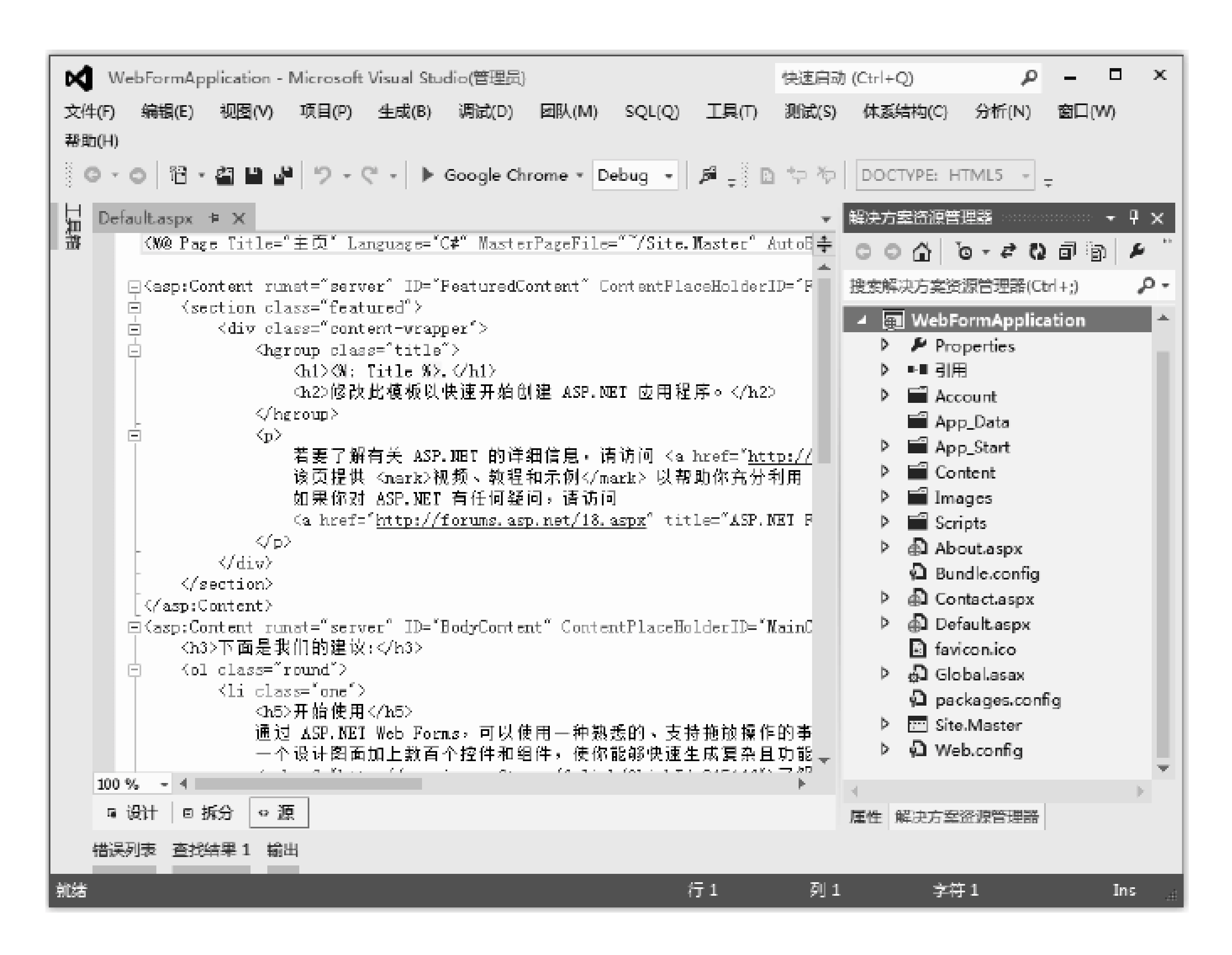


图 2-1　基于窗体的 Web 应用程序

（3）App_Data 目录：包含 Microsoft Office Access 和 SQL Expression 文件以及 XML 文件或者其他数据存储文件。

（4）Images 目录：包含图像文件。

（5）Scripts 目录：包含脚本文件。

（6）Global.asax 文件：这是一个可选文件，通常被称为 ASP.NET 应用程序文件。该文件包含响应 ASP.NET 或 HTTP 模块所引发的应用程序级别和会话级别事件的代码。如果文件不存在进行创建时，必须将其放在应用程序和根目录下。

（7）Web.config 文件：该文件用来存储 ASP.NET Web 应用程序的配置信息，这是一个 XML 文件。

在如图 2-1 所示界面中，Content、Images、Scripts、Abount.aspx 等目录和文件都是基于窗体的 Web 应用程序生成的。开发者也可以创建不带窗体的 Web 应用程序，创建时只需要在弹出的对话框中选择【ASP.NET 空 Web 应用程序】项即可。如图 2-2 所示为创建空 Web 应用程序时的结构。

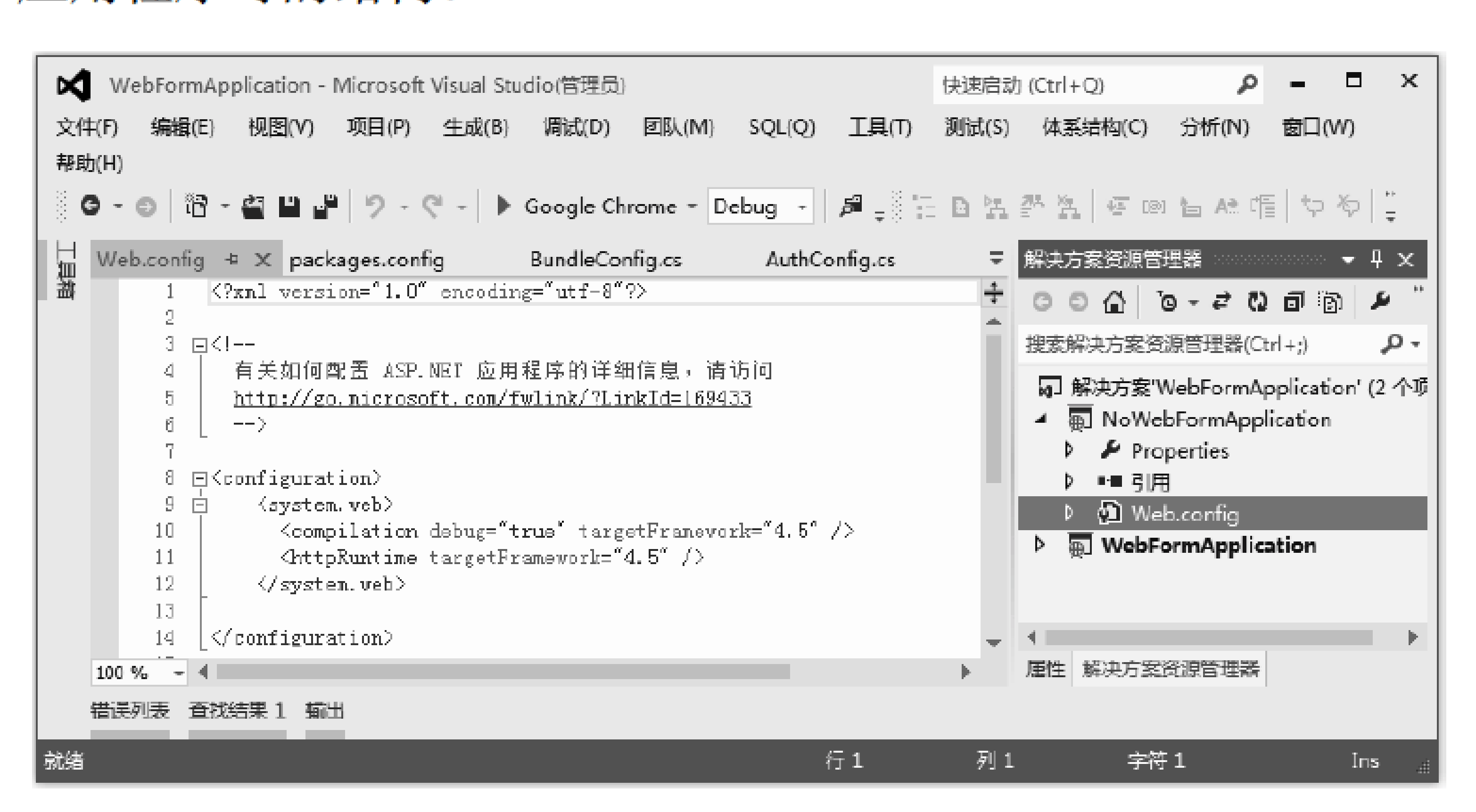


图 2-2　不带窗体的 Web 应用程序

比较图 2-1 和图 2-2 可以看出，不带窗体的 Web 应用程序很简单，只包含 Properties 目录、引用文件目录和 Web.config 文件。

2.1.2　新建 Web 网站

开发者可以通过创建 Web 窗体应用程序的方式创建 Web 程序，还可以通过创建 Web 网站的方式创建 Web 程序。选择【文件】|【新建】|【网站】命令，打开【新建网站】对话框，如图 2-3 所示。



图 2-3　基于窗体的 Web 网站

在如图 2-3 所示界面中，选择【ASP.NET Web 窗体网站】后输入或选择网站位置，然后单击【确定】按钮创建基于窗体的 Web 网站，如图 2-4 所示。

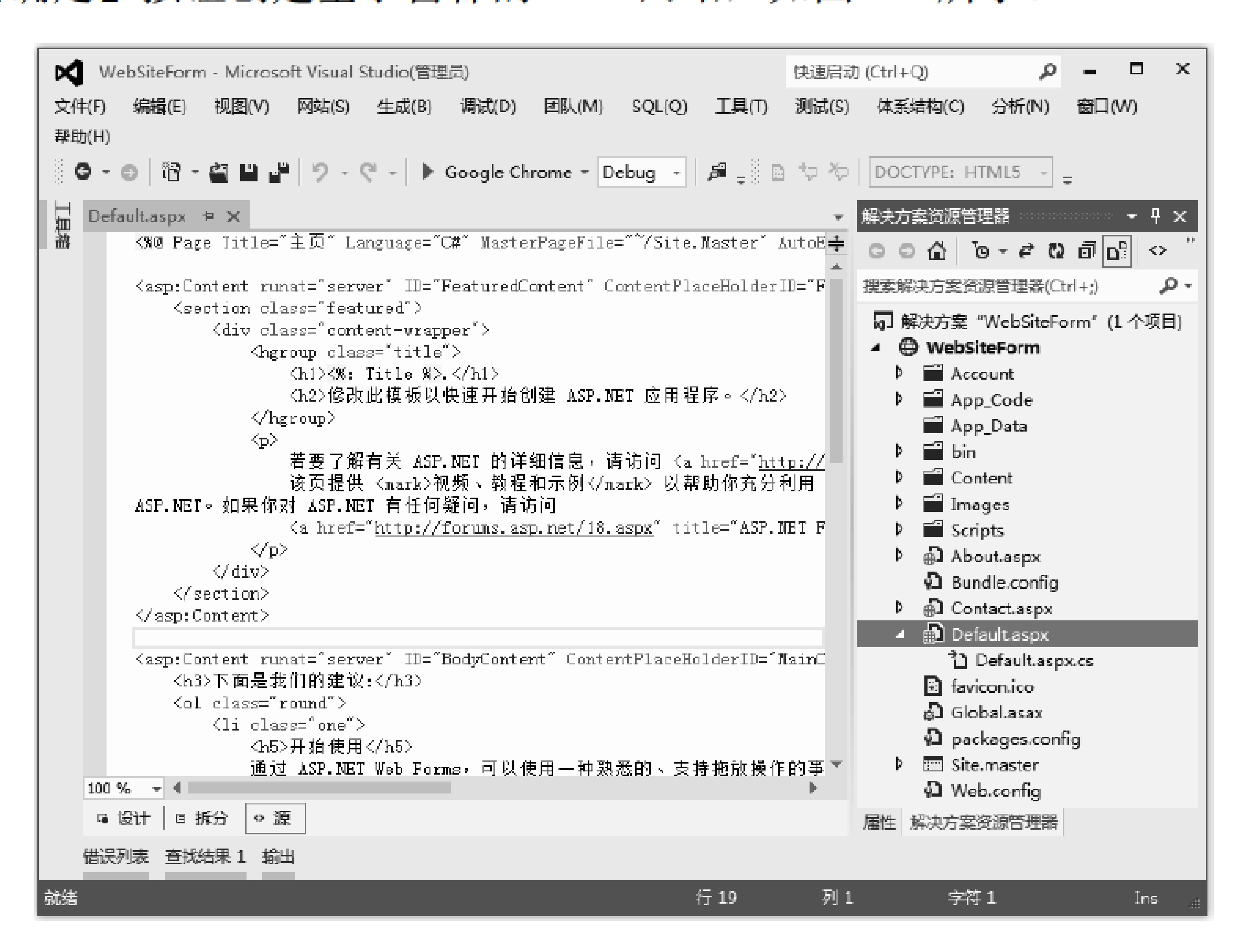


图 2-4　基于窗体的 Web 网站

在如图 2-4 所示界面中，基于窗体的 Web 网站也会生成多个目录和文件，其中许多目录和文件的说明都与基于窗体的 Web 应用程序相似，这里只介绍 App_Code 和 bin 目录。

（1）App_Code 目录：包含作为应用程序的一部分编译的类的源文件。当页面被请求时，ASP.NET 编译该目录中的代码，该目录中的代码在应用程序中自动地被引用。

（2）bin 目录：包含应用程序所需的任何预生成的程序集。

开发者也可以创建不带窗体的 Web 网站，在如图 2-3 所示的对话框中选择【ASP.NET 空网站】即可。如图 2-5 所示为不带窗体的 Web 网站结构。从图 2-5 中可以看出，创建空网站时只生成一个 Web.config 文件。



图 2-5　不带窗体的 **Web** 网站结构

2.1.3　比较 Web 应用程序和 Web 网站

VS 2012 中既可以创建 Web 应用程序，也可以创建 Web 网站。一般来说，Web 应用程序适合相对较大的系统，而 Web 网站比较适合中小型企业网站。下面分别从它们的相同点和不同点两方面进行说明。

1．相同点

Web 应用程序和 Web 网站有两个相同点：都用来设计和实现 ASP.NET 网页；都可以添加 ASP.NET 文件夹，如都包括 App_Browsers、App_Data、App_GlobalResources、App_LocalResources 和 App_Themes。

2．不同点

Web 应用程序和 Web 网站存在多个不同点，说明如下。

（1）Web 应用程序 Default.aspx 显示有两个原有文件，分别是 Default.aspx.cs 和 Default.aspx.designer.cs；Web 网站 Default.aspx 有一个原有文件，即 Default.aspx.cs。

（2）Web 应用程序有重新生成和发布网站两项；Web 网站只有一个发布网站。

（3）Web 应用程序和一般的 WinForm 没有什么区别，如都引用的是命名空间；Web 网站在引用后出现一个 bin 目录，该目录存在后缀名为“.dll”和“.pdb”的文件。

（4）Web 应用程序可以作为类库被引用；Web 网站则不可以作为类库被引用。

（5）Web 应用程序可以添加 ASP.NET 文件夹，但是不包括 bin 和 App_Code；Web 网站可以添加 ASP.NET 文件夹，但是包括 bin 和 App_Code。

（6）Web 应用程序可以添加组件和类；Web 网站则不能。

（7）源文件虽然都是 Default.aspx.cs 文件，但是 Web 应用程序多了对 System.Collections 命名空间的引用。

2.2 Web 窗体页

无论是创建 Web 应用程序还是 Web 网站，都可以为其添加 Web 窗体页。使用 Web 窗体页创建可编程的 Web 页面，这些 Web 页面用作 Web 应用程序的用户界面。Web 窗体页可以在任何浏览器或者客户端设备中向用户提供信息，并使用服务器端代码来实现应用程序逻辑。

Web 窗体页几乎可以包含任何支持 HTTP 的语言，如 HTML 与 XML、WML、JScript 和 JavaScript 等。

2.2.1 Web 窗体页的特点

Web 窗体页基于 ASP.NET 技术，它是创建 ASP.NET 网站和 Web 应用程序编程模式常用的一种。Web 窗体页是整合了 HTML、服务器控件和服务器代码的事件驱动网页，特点如下。

（1）兼容所有的浏览器和设备。Web 窗体页自动为样式、布局等功能呈现正确的、符合浏览器的 HTML。开发者还可以选择将 Web 窗体页设计为在特定浏览器（如 IE 5）上运行并利用多样式浏览器客户端的功能。

（2）兼容.NET 公共语言运行时所支持的任何语言，其中包括 Visual Basic、C#和 JScript。

（3）基于 Microsoft .NET Framework 生成，它提供了该框架的所有优点，包括托管环境、类型安全性和继承。

（4）在 VS 中为快速应用程序开发提供支持，该工具用于对窗体进行设计和编程。

（5）可使用为 Web 开发提供应用程序功能的控件进行扩展，从而使开发者能够快速地创建多样式的用户界面。

（6）具有灵活性，开发者可以添加用户创建的控件和第三方控件。

2.2.2 Web 窗体页的元素

在 Web 窗体页中，用户界面编程被分为两个不同的部分：可视元素和逻辑。可视元素称作 Web 窗体“页”，这种页是由一个包含静态 HTML 和/或 ASP.NET 服务器控件的文件组成的。Web 窗体页的逻辑由代码组成，这些代码由开发者创建与窗体进行交互。

针对可视元素和逻辑，ASP.NET 提供两个用于管理它们的模型，即单文件页模型和

代码隐藏页模型。这两个模型功能相同，两种模型中可以使用相同的控件和代码。

1. 单文件页模型

在单文件页模型中，页的标记及其编程代码位于一个物理文件（即.aspx 文件）中，编程代码位于脚本块中，该块包含 runat=server 属性，该属性标记该块或控件在服务器端执行。

使用单文件页模型有以下几个优点。

（1）可以方便地将代码和标记保留在同一个文件中。

（2）更容易部署或发送给其他程序员。

（3）由于文件之间没有相关性，更容易对单文件页进行重命名。

（4）更易于管理源码文件。

2. 代码隐藏页模型

通过代码隐藏页模型，可以在一个文件（即.aspx 文件）中保留标记，并在另一个文件（即.aspx.cs 文件）中保留编程代码，该文件被称为“代码隐藏”文件或“页面后台”文件，代码文件的名称会根据所使用的编程语言而有所变化。

使用代码隐藏页模型包括以下几个优点。

（1）代码隐藏页可以清晰地区分界面中的标记控件和程序代码。

（2）代码并不会向界面设计人员或其他人员公开。

（3）代码可以在多个页面中进行重用。

2.2.3 Web 窗体页的运行过程

当 ASP.NET Web 应用程序运行时，每一个被请求的 Web 窗体页都将经历一个运行过程，即生命周期。在该运行过程中，ASP.NET 将对 Web 窗体页进行一系列的处理，如页请求、初始化页面、载入页面、处理事件、预呈现页面、呈现页面和卸载页面等。一个 Web 窗体从实例化分配内存空间到处理结束释放内存，大体分为 4 个步骤：页面初始化、页面装载、事件处理和资源清理。

1. 页面初始化

当页面被初始化时，发生第一个 Page_Init 事件，系统会执行创建和设置一个示例所需要的所有初始化步骤。

2. 页面装载

页面装载在初始化之后，所引发的是 Page_Load 事件，用途如下。

（1）根据 Page.IsPostBack 属性检查页面是不是每一次被处理。

（2）每一次处理页面时执行数据绑定，或者在以后的循环过程中重新判断数据绑定表达式。

（3）读取和更新控件属性。

（4）恢复在保存步骤中所保存的前一个客户请求的状态。

3．事件处理

Web 窗体上的每个动作都激活一个到达服务器的事件。一个 Web 窗体有两个视图：一个客户视图和一个服务器视图。所有的数据处理都在服务器上进行，当通过单击鼠标或者其他方法引起一个事件时，事件就到达服务器并返回相应的数据。

4．资源清理

最后一步发生在一个窗体完成了它的任务并且准备卸载的时候，这时激活 Page_Unload 事件，完成最后的资源清理工作，如关闭文件、关闭数据库连接和丢弃对象等。

2.2.4 认识 Web 窗体页

Web 窗体页以“.aspx”结尾，当创建基于窗体的 Web 应用程序或网站时会自动生成一些 Web 窗体页。当然，开发者也可以亲自动手创建 Web 窗体页。

【范例 1】

首先创建一个基于窗体的 Web 应用程序，然后在该程序中创建全称是 WorkDefault.aspx 的 Web 窗体页。创建完毕后打开页面，【源】窗口中的代码如图 2-6 所示。

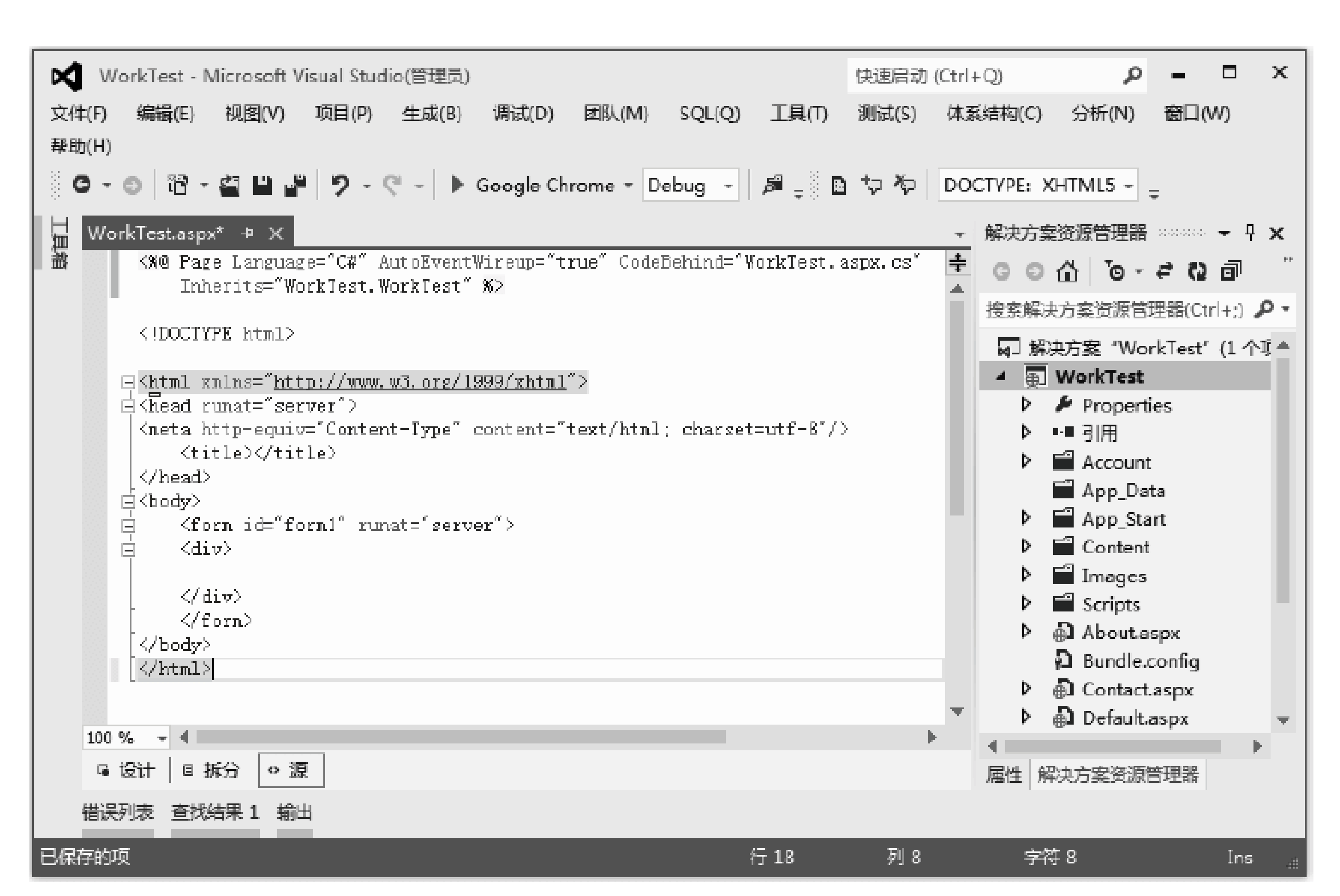


图 2-6 Web 窗体页【源】窗口代码

在如图 2-6 所示界面中，单击图中的【设计】标签可打开【设计】窗口，单击【拆分】标签可查看页面的源代码和设计效果。观察图 2-6 中的代码，首先通过@Page 指令定义 ASP.NET 页解析器和编译器所使用的特定的页面的属性，然后添加一段 HTML 代码。开发者可以在 body 元素中添加 HTML 服务器控件、Web 服务器控件或其他内容。

运行页面很简单，在该页面右击，然后选择【在浏览器中查看】项即可。或者选中程序中的 Web 窗体页右击，然后选择【在浏览器中查看】项运行。

2.3 ASP.NET 的页面指令

ASP.NET 页面中通常包含一些类似<%@ %>的代码，被称为页面指令。这些指令允许指定一些属性和配置信息，就是 ASP.NET 用作处理页面的指令。当使用指令时，标准的做法是将指令包括在文件的开头，也可以将它们置于“.aspx”或“.ascx”文件中的任何位置。本节介绍 ASP.NET 中提供的一些基本指令，每个指令都可以包含一个或者多个特定于该指令的属性。

2.3.1 @Page 指令

指令用以控制页面生成，它通常有两个作用：指定页面在处理 Web 窗体页文件（扩展名为“.aspx”的文件）时使用哪些设置；指定用户控件编译器在处理用户自定义控件（扩展名为“.ascx”）时使用哪些设置。

@Page 指令定义 ASP.NET 页解析器和编译器所使用的特定的页面属性。@Page 指令只能在“.aspx”页面中使用，如果在其他 ASP.NET 页（如控件和 Web 服务）中使用会导致编译错误。每一个“.aspx”文件最多只能包含一个@Page 指令，基本语法如下。

```
<%@ Page attribute="value" [attribute="value"…]%>
```

@Page 指令大约有三十个属性，它们从逻辑上分为三类：编译、页面整体行为和页面输出。如表 2-1 所示只对@Page 指令的常用属性进行说明。

表 2-1 @Page 指令的常用属性

属性名称	说明
Language	用于在编译时提示内联代码块（<%%>）和<script>区段中代码所使用的语言
AutoEventWireup	布尔类型的属性，用于指示是否启动页面的事件，默认值为 true
CodeBehind	用于提示当前页面代码隐藏类的路径
Inherits	用于定义当前页要继承的基类，它可以为从 Page 类派生的任何类
EnableViewState	布尔类型的属性，用于指示是否在页面请求间保持视图状态
EnableTheming	布尔类型的属性，用于指定当前页是否对嵌入的控件应用主题
MasterPageFile	用于指示当前页面的母版页
Src	用于指示包含实现 Inherits 指定的基类的源文件路径

2.3.2 @Control 指令

@Control 指令类似于@Page 指令，但是它定义 ASP.NET 页解析器和编译器所使用的特定的用户控件的属性。@Control 指令只能用在用户控件中，每一个“.ascx”文件最多只能包含一条@Control 指令。基本语法如下。

```
<%@ Control attribute="value" [attribute="value"…]%>
```

@Control 指令的可用属性要比@Page 指令少，但是其中许多属性都可以在创建用户控件时根据需要进行修改。创建用户控件时会自动添加@Control 指令，代码如下。

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="
WebUserControl1.ascx.cs" Inherits="WorkTest.WebUserControl1" %>
```

2.3.3 @Register 指令

@Register 指令用于注册页面中的用户控件和自定义服务器控件以使用它们。把一个用户控件拖放到“.aspx”页面上时，就会使用到@Register 指令；把一个第三方控件拖放到“.aspx”页面上时，也会使用到@Register 指令。这样，在页面上注册用户控件或第三方控件之后就可以在“.aspx”页面上访问该控件了。

@Register 指令支持 5 个属性，如表 2-2 所示。

表 2-2 @Register 指令的常用属性

属性名称	说明
Assembly	与 TagPrefix 关联的命名空间所驻留的程序集
Namespace	与 TagPrefix 关联的命名空间
Src	用户控件文件的位置
TagName	与类关联的别名
TagPrefix	与命名空间关联的命名

2.3.4 @Master 指令

@Master 指令用于指示当前页面标识为 ASP.NET 母版页。简单来说，@Master 指令用于创建母版页。在使用@Master 指令时，要指定和站点上的内容页面一起使用的模板页面的属性，内容页面（使用@Page 指令创建）可以继承母版页上的所有内容。

@Master 指令与@Page 指令类似，但是它的属性要比@Page 指令的属性少。创建一个母版页时生成的@Master 指令的代码如下。

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.
cs" Inherits="WorkTest.SiteMaster" %>
```

2.3.5 @MasterType 指令

@MasterType 指令把一个类名关联到 ASP.NET 页面上，以获取特定母版页中包含的强类型化的引用或成员。@MasterType 指令支持 TypeName 和 VirtualPath 两个属性。

（1）TypeName 属性：设置从中获取强类型化的引用或成员的派生类的名称。

（2）VirtualPath 属性：设置从中检索这些强类型化的引用或成员的页面地址。

如下代码简单演示了@MasterType 指令的使用。

```
<%@ MasterType VirtualPath="~/MyWork.master" %>
```

2.3.6 @Import 指令

@Import 指令在页面或用户控件中显式地引入一个命名空间，以便所有已定义类型可以在页面访问，而不必使用完全限定名。例如，创建 ASP.NET 中 DataSet 类的实例时，可以导入 System.Data 命名空间，也可以使用完全限定名。使用完全限定名的代码如下。

```
System.Data.DataSet ds = new System.Data.DataSet();
```

@Import 指令可以在页面主体中多次使用，它相当于 C#中的 using 语句。如下为它的简单例子。

```
<%@ Import Namespace="System.Data" %>
```

2.3.7 @Implements 指令

@Implements 指令允许在页面或用户控件中实现一个.NET Framework 接口，该指令只支持 Interface 属性。Interface 属性直接指定.NET Framework 接口。当 ASP.NET 页面或用户控件实现接口时，可以直接访问其中所有的事件、方法和事件。

如下是使用@Implements 指令的例子。

```
<%@ Implements Interface="System.Web.UI.IValidator" %>
```

2.3.8 @Reference 指令

@Reference 指令用来识别当前页面在运行时应该动态编译和链接的页面或用户控件。该指令在跨页通信方面发挥着重大作用，可以通过它来创建用户控件的强类型实例。

@Reference 指令可以多次出现在页面中，并且包含三个属性：Page、Control 和 VirtualPath。

（1）Page 属性：该属性用于指向某个“.aspx”源文件。

（2）Control 属性：该属性包含“.ascx”用户控件的路径。

（3）VirtualPath 属性：设置从中引用活动页面的页面或用户控件的位置。

2.3.9 @Assembly 指令

@Assembly 指令将程序集引入到当前页面或用户控件中，以便它所包含的类和接口能够适用于页面中的代码。@Assembly 指令支持 Name 和 Src 两个属性。

（1）Name 属性：允许指定用于关联页面文件的程序集名称。程序集名称应只包含文件名，不包含文件的扩展名。假设文件是 MyAssembly.vb，那么 Name 属性的值是 MyAssembly。

（2）Src 属性：允许指定编译时所使用的程序集文件源。

2.3.10 @OutputCache 指令

@OutputCache 指令对页面或用户控件在服务器上如何输出高速缓存进行控制。该指令常用属性如表 2-3 所示。

表 2-3 @OutputCache 指令的常用属性

属性名称	说明
Duration	ASP.NET 页面或用户控件高速缓存的持续时间。单位为 s
CacheProfile	允许使用集中式方法管理应用程序的调整缓存配置文件。该属性指定在 Web.config 文件中详细说明的调整缓存配置文件名
Location	位置枚举值，只对“.aspx”页面有效，不能用于用户控件。其值包括 Any（默认值）、Client、Downstream、None、Server 和 ServerAndClient
NoStore	指定是否随页面发送没有存储的标题
Shared	指定用户控件的输出是否可以在多个页面中共享。默认值为 false
SqlDependency	支持页面使用 SQL Server 高速缓存禁用功能
VeryByControl	用分号分隔开的字符串列表，用于改变用户控件的输出高速缓存
VeryByParam	用分号分隔开的字符串列表，用于改变输出高速缓存
VeryByCustom	一个字符串，指定定制的输出高速缓存需求
VerByHeader	用分号分隔开的 HTTP 标题列表，用于改变输出高速缓存

2.3.11 @PreviousPageType 指令

@PreviousPageType 指令用于指定跨页面的传送过程起始于哪个页面。@PreviousPageType 是一个新指令，用于处理 ASP.NET 提供的跨页面传送新功能。这个简单的指令只包含两个属性：TypeName 和 VirtualPath。

（1）TypeName 属性：设置回送时的派生类的名称。

（2）VirtualPath 属性：设置回送时所传送页面的地址。

注 意

在前面介绍的指令中，除了@Page、@Control、@Master、@MasterType 和@PreviousPageType 外，所有指令都可以在页面和控件中声明。@Page 和@Control 是互斥的：@Page 只能用在“.aspx”文件中；@Control 只能用在“.ascx”文件中。

思考与练习

一、填空题

1. 针对 Web 窗体页中的可视元素和逻辑，ASP.NET 提供单文件页模型和__________来管理它们。

2. Web 窗体页的运行过程可分为页面初始化、页面装载、__________和资源清理 4 个步骤。

3. __________指令定义 ASP.NET 页解析器

和编译器所使用的特定的页面属性。

4. @PreviousPageType 指令的常用两个属性是__________和 VirtualPath。

二、选择题

1. 在创建基于窗体的 Web 应用程序时，生成的__________目录包含 Microsoft Office Access 和 SQL Expression 文件以及 XML 文件或者其他数据存储文件。

A. Properties

B. Account

C. App_Themes

D. App_Data

2. 使用代码隐藏页模型的优点不包括__________。

A. 可以清晰地区分界面中的标记控件和程序代码

B. 代码并不会向界面设计人员或其他人员公开

C. 可以方便地将代码和标记保留在同一个文件中

D. 代码可以在多个页面中进行重用

3. __________指令只能用在用户控件中，每一个用户控件中最多只能包含一条该指令。

A. @Control

B. @Page

C. @Master

D. @Import

4. @Register 指令的__________属性指定与类关联的别名。

A. TagName

B. TagPrefix

C. Namespace

D. Assembly

5. @Reference 指令的属性不包括________。

A. Page

B. Duration

C. Control

D. VirtualPath

三、简答题

1. 简单概述 Web 应用程序和 Web 网站的异同点。

2. Web 窗体页的运行过程是什么？请简单说明。

3. ASP.NET 的页面指令有哪些？这些指令分别用来做什么？

第 3 章　Web 服务器控件

控件是一种具有特殊作用的对象，在 ASP.NET 中，一切都由对象组成。Web 页面就是一个对象的容器，而控件是 Web 页面的元素之一。常见的控件有按钮、复选框、下拉框、图片等，直接在页面中展示给用户。

了解这些控件的属性、方法和事件对以后的学习尤其重要。本章详细介绍服务器控件的类型、共有属性及控件和页面的综合应用。

本章学习要点：

- ❑ 了解服务器控件的特点
- ❑ 理解服务器控件的公共属性
- ❑ 掌握文本控件的使用
- ❑ 掌握按钮控件的使用
- ❑ 掌握复选框的使用
- ❑ 掌握列表控件的使用
- ❑ 掌握容器控件的使用
- ❑ 掌握控件与页面的结合

3.1 服务器控件基础

HTML 中也有控件的概念，并实现指定的应用。但服务器控件拥有更好的数据处理方法和响应事件，实现更为复杂的功能。本节介绍服务器控件的基础知识，包括服务器控件概述、分类和公共属性等。

3.1.1 服务器控件概述

服务器控件是需要在服务器端响应的控件。与 HTML 控件相比，服务器控件有着更为便利的处理数据的方法和与用户交互的响应事件。

例如，服务器控件又有数据的回送功能，在页面刷新或关闭时回送数据，以确保重要的数据被记录下来。除此之外，服务器控件还有以下几个特点。

（1）服务器控件可以触发服务器控件特有的事件。

（2）输入到服务器控件中的数据在请求之间可以维护（即具有状态管理功能）。

（3）服务器控件可以自动检测浏览器并调整到恰当的显示。

（4）每个服务器控件都具有一组属性，可以在服务器端的代码中更改控件的外观和行为。

ASP.NET 服务器控件可以自动检测客户端浏览器的类型，产生一个或者多个适当的

HTML 控件，并且自动调整成适合浏览器的输出。服务器控件支持数据绑定技术，可以和数据源进行连接，用来显示或修改数据源数据。其优点如下所示。

（1）使制造商和开发人员能够生成容易的工具或者自动生成用户的应用程序接口。

（2）简化创建交互式 Web 窗体的过程。

将服务器控件添加到 Web 窗体中非常简单，最常用的方法有三种，如下所示。

（1）从工具箱中拖动控件到窗体上或直接双击控件进行添加。

（2）在资源视图中，直接添加控件的声明代码。

（3）以编程方式动态创建 Web 服务器控件。

注 意

所有的 ASP.NET 控件必须定义在.aspx（ASP.NET 页面文件）文件中，如果是采用代码隐藏技术设计的程序，其事件程序一般定义在代码文件（如.cs 或.vb）中。

3.1.2 服务器控件分类

服务器控件种类繁多，根据控件所实现的作用可将控件分为标准控件、数据控件、验证控件、导航控件和登录控件，对其介绍如下。

1. 标准控件

标准控件是所有控件中最为常用的，有按钮控件、文本控件、复选框控件和容器控件等。而仅文本相关的控件就有文本输入框、文本显示标签和链接文本等控件。根据控件的样式可以将标准控件分为文本控件、按钮控件、选择控件、列表控件和容器控件。

2. 数据控件

数据控件主要用于数据的处理，包括数据的绑定和显示。ASP.NET 页面中的数据通常是动态数据，需要不断变化。这些数据被存放在数据源文件中，数据显示控件在绑定了数据源之后，所显示的数据会随着数据源数据的变化而变化。

除此之外，数据控件也可以操作数据源中的数据。ASP.NET 页面通常与数据库结合，以数据库作为数据源来存储数据。数据控件的使用将在第 10 章中介绍。

3. 验证控件

网站通常是需要与用户“交流”的，如用户在登录时需要提供用户名和密码、在查询时需要提供查询关键字、在注册的时候需要提交基本信息等。

用户与网站之间的“交流”表现在信息的输入输出方面，但是在大多情况下，网站对用户的输入格式会有限制，如用户注册时的邮箱地址不合法，将直接影响用户的注册。为了确保用户输入的格式符合网站需求，服务器控件中提供了验证控件对用户的输入进行验证，只有通过了验证的输入才能被网站接收处理。

验证控件通常与输入文本框进行绑定，以验证用户的输入是否有效。验证控件将在第 8 章中介绍。

4．导航控件

导航控件与网站用户的关系，相当于导游和旅行者的关系。导航控件能够将大型的网站系统结构化，列出网页之间的逻辑关系，使用和能够方便快捷地找到所需的页面进行访问。导航控件通常和站点地图等列举页面逻辑的文件结合使用，在第 6 章中将详细介绍。

5．登录控件

用户登录通常需要输入用户名、密码，并由网站对用户名和密码的合法性进行验证，在验证无误的情况下提交信息。

由于登录功能是网站的常用功能，因此 ASP.NET 将登录常用的控件定义为一个组合，可直接从工具箱拉入页面使用。

登录控件通常包含【用户名】文本框、【密码】文本框等关于登录的常见控件。登录控件的使用减轻了开发人员的工作。

3.1.3 服务器控件公共属性

ASP.NET 服务器控件有多种类型，但这些控件有着它们共同的特点和属性。每一个 Web 服务器控件都有一个<asp:>的前缀，该前缀表示此控件为 Web 服务器控件。其语法格式如下。

```
<asp:Control id="name" runat="server" />
```

上述语法中，id 表示控件的唯一标识，runat 属性指示该控件为服务器控件，这两个属性的含义与 HTML 控件中相应的属性的含义一致。例如，下面的代码表示在 ASP.NET 页面中添加一个 Label 控件。

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

ASP.NET 中的服务器控件继承自 System.Web.UI.WebControls 类，该类提供了大多数 Web 服务器控件的公共属性、方法和事件。如表 3-1 所示对服务器控件的最常用的一些属性进行了说明。

表 3-1 服务器控件的常用属性

属性名称	说明
AccessKey	获取或设置快速导航到控件的快捷键，可以指定这个属性的内容为数字或者是英文字母
BackColor	设置对象的背景颜色，其属性的值可以是颜色名称，也可以是#RRGGBB 格式
Enabled	获取或设置一个值，该值指示是否启用 Web 服务器控件
Visible	指定控件是否可见
ToolTip	小提示。在设置该属性时，当使用者停留在 Web 控件上时就会出现提示的文字
ID	所有控件的唯一标识列
FailureText	获取或设置当前登录尝试失败时显示的文本

除了表 3-1 中的内容，服务器控件都有 runat="server"属性值，以区分服务器控件和 HTML 控件。设置控件的属性时有以下两种方法。

（1）在设计时通过【属性】窗格设置控件的属性。

（2）在运行时以编程方式动态设计控件的属性。

服务器控件是显示在页面中与用户进行交互的，因此控件通常能够因为用户的操作而触发某些事件的执行。如用户单击按钮，那么按钮的鼠标单击事件将被执行；用户修改文本框中的内容，那么若该文本框定义了修改事件，该事件将被执行。

事件是一种在满足某种条件（如鼠标单击）后开始运行的程序，大部分的 ASP.NET 控件都可以引发服务器端事件完成某些功能，页面事件是在页面加载时和撤销时所引发的事件。

页面级别的事件主要有三种：Page_Load、Page_Init 和 Page_Unload。Page_Init 事件和 Page_Load 事件都是在页面加载时引发并用来执行初始化程序的事件，前者只是在页面第一次加载时执行的事件，而 Page_Load 事件在每次加载时都执行。Page_Unload 事件执行最后的清理工作，例如关闭打开的文件和数据库连接等。

向 ASP.NET 服务器控件添加客户端事件时有三种方法，如下所示。

（1）以声明方式向 ASP.NET 服务器控件添加客户端事件处理程序，即在资源视图中直接为控件添加事件属性，如 onmouseover 或 onclick 等。添加事件属性时需要针对不同的属性添加要执行的客户端脚本。

（2）以编程方式向 ASP.NET 控件添加客户端事件处理程序，即在页面的 Init 或 Load 事件中调用控件的 Attributes 集合的 Add()方法来动态添加客户端事件处理程序。

（3）向按钮控件添加客户端 onclick 事件。例如，在按钮控件（Button、LinkButton 和 ImageButton）中要添加客户端 onclick 事件，可以在设计视图中将按钮控件的 onclientclick 属性设置为要执行的客户端脚本，也可以在源视图中直接添加该属性。

3.2 文本控件

文本控件是指文本相关的控件，包括文本的显示和输出。本节主要介绍文本控件的适用范围和使用方法。

3.2.1 Label 控件

Label 的英文含义是标签，其在页面中的作用正如一个可以被写入文字的标签。Label 控件也称作标签控件，其在页面中的显示是一行（或一段）文字。

Label 控件的添加可以在工具箱中进行拖拉，其在工具箱中的符号为**A** Label。该控件的 Text 属性值，是页面中所显示的文本。

该属性的值可以是文本，也可以是浏览器能够识别的页面元素，如在文本当中含有
元素，则文本的显示会有换行。

Label 提供了一种以编程方式设置文本的方法，可以在后台对文字的内容和样式进行设置。

Label 控件中的文本可以是静态的和动态的，静态的情况下在页面固定位置显示文本；动态情况下可以绑定到数据源。声明 Label 控件有两种语法形式，如下所示。

```
<asp:Label ID= "lblName" runat = "server" Text = "文本内容"></asp:Label>
//或者
<asp:Label ID= "lblName" runat = "server">文本内容</asp:Label>
```

直接从工具箱中拖出的 Label 控件，其代码如下所示。

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

除了 runat="server"属性以外，Label 控件默认含有 ID 和 Text 属性。其中，ID 用来设置控件的唯一标识列；Text 属性向用户显示文本信息，该属性的值可以是 HTML 格式的内容。

ID 属性是不可缺少的身份识别，若同时存在多个 Label 控件，该值是不可缺少的；而 Label 控件的作用即为显示文本信息，因此 Text 属性的默认值将使其失去意义。

Label 控件的 Text 属性可以设置为任何字符串（包括包含标记的字符串），如果字符串包含标记，Label 控件将解释标记。Label 控件还提供了一系列的样式属性，以设置其文本的展示样式，如字体的颜色、字体背景色大小、下划线、是否加粗等，如设置一个红色加粗的标签，代码如下。


```
<asp:Label  ID="Label1"  runat="server"  Text="Label"  Font-Bold="True"
ForeColor="Red"></asp:Label>
```

3.2.2 Literal 控件

Literal 控件的作用同样是在页面中显示文本，但 Literal 控件不允许对所显示的文本应用样式。

在工具箱中，Literal 控件的符号为 Literal。Literal 控件与 Label 控件一样用于显示文本，但 Literal 控件可以在浏览器中进行动态的添加，通常将其添加到指定容器内。

Literal 控件与 Label 控件最大的区别在于：Literal 控件不能向文本中添加任何的 HTML 元素。因此，Literal 控件不支持包括位置特性在内的任何样式特性。但是，该控件允许对其内容进行编码。

通常情况下，如果开发人员希望文本和控件直接呈现在页面中而不使用任何附加标记时可以使用 Literal 控件。除了 ID 和 Text 属性外，Literal 控件最常用的属性是 Mode，该属性用于指定控件对所添加的标记的处理方式。其值如下所示。

（1）Transform：将对添加到控件中的任何标记进行转换，以适应请求浏览器的协议。如果向使用 HTML 外的其他协议的移动设备呈现内容，设置该值时非常有用。该值为 Mode 属性的默认值。

（2）PassThrough：添加到控件中的任何标记都将按原样呈现在浏览器中。

（3）Encode：使用 HtmlEncode()方法对添加到控件中的任何标记进行编码，这会将 HTML 编码转换为其文本表示形式。例如，<b>将呈现为<b>。当开发人员希望浏

览器显示而不解释标记时，该方式将非常有用。

如向页面中添加 Literal 控件，使其 ID 为“Literal”；显示文本为“动态添加”；将控件添加在 form1 中，使用代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    Literal liter = new Literal();           //通过 Literal 创建 liter 对象
    liter.ID = "Literal";                    //设置 ID 属性
    liter.Text = "动态添加";                  //设置 Text 属性
    form1.Controls.Add(liter);               //将 liter 添加到表单中
}
```

上述代码中，form1 作为一个容器，接收 Literal 控件的添加。但这个添加过程无法设置 Literal 控件的添加位置。

3.2.3 TextBox 控件

Label 控件和 Literal 控件属于文本显示控件，而 TextBox 控件属于文本输入控件。这三种控件都有 Text 属性，该属性可设置 Label 和 Literal 控件的显示文本，也可以获取用户在 TextBox 上面输入的文本。

TextBox 控件又称作文本框控件，其在工具箱中的符号为 abl TextBox。TextBox 控件提供一个输入框，供用户输入。文本框中输入的值即为该控件的 Text 属性值，可在后台通过控件的 ID 进行获取。接下来对控件的属性、事件和自动完成功能进行介绍。

1. TextBox 控件的常用属性

TextBox 控件是一种基本控件，它为用户提供了一种在 Web 窗体中输入信息（包括文本、数字和日期等）的方法。该控件的常用属性如表 3-2 所示。

表 3-2 **TextBox 控件的常用属性**

属性名称	说明
AutoPostBack	获取或设置当 TextBox 控件上的内容发生改变时，是否自动将窗体数据回传到服务器，默认为 false。该属性通常和 TextChanged 事件配合使用
MaxLength	获取或设置文本框中最多允许的字符数。当 TextMode 属性设为 MultiLine 时，此属性不可用
ReadOnly	获取或设置 TextBox 控件是否为只读。默认值为 false
TextMode	获取或设置文本框的行为模式。默认值为 SingleLine
Wrap	布尔值，指定文本是否换行。默认为 true

TextBox 控件可以通过 TextMode 属性设置出单行、多行和密码三种形式的文本框。TextMode 属性的可取值如下。

（1）SingleLine：默认值，单行输入模式。用户只能在一行中输入信息，还可以限制控件接受的字符数。

（2）Password：密码框，用户输入的内容将以其他字符代替（如*和●等），以隐藏

真实信息。

（3）Multiline：多行输入模式，用户在显示多行并允许换行的框中输入信息。

开发人员可以通过 Text 属性获取或设置 TextBox 控件中的值，例如，获取控件 ID 为 TextBox1 的属性值并将其赋予字符串变量 name，代码如下。

```
string name =TextBox1.Text;
```

2．TextBox 控件的事件

TextBox 控件最常用的事件是 TextChanged 事件，当用户离开 TextBox 控件时就会引发该事件。默认情况下并不会立即引发该事件，而是当下次发送窗体时在服务器代码中引发此事件。控制这一事件的属性是 AutoPostBack，该属性的默认值为 false。将该属性的值设置为 true，用户离开 TextBox 控件后将页面提交给服务器。

例如，为 TextBox 控件添加 TextChanged 事件，当用户离开 TextBox1 文本框后触发该事件将值赋予 Label 控件进行显示，代码如下。

```
protected void TextBox1 TextChanged(object sender, EventArgs e)
{
    Label1.Text = Server.HtmlEncode(TextBox1.Text);
}
```


3．TextBox 控件的自动完成功能

许多浏览器都支持自动完成功能，该功能可以帮助用户根据以前输入的值向文本框中填充信息。自动完成的精确行为取决于浏览器，通常浏览器根据文本框的 name 特性存储值；任何同名的文本框（即使是在不同页上）都将为用户提供相同的值。

有些浏览器还支持 vCard 架构，该架构允许用户使用预定义的名、姓、电话号码和电子邮件地址等值在浏览器中创建配置文件。

TextBox 控件支持 AutoCompleteType 属性，该属性为用户提供了用于控制浏览器如何使用自动完成的选项。

3.2.4　HyperLink 控件

在文本的显示中，还有一个特殊的控件 HyperLink 控件。HyperLink 控件用于显示文本和图片的超链接，其在工具箱中的符号为A　HyperLink。

与大多数 Web 服务器控件不同，当用户单击 HyperLink 控件时并不会在服务器代码中引发事件，该控件只执行导航操作。HyperLink 控件有以下两个优点。

（1）可以在服务器代码中设置链接属性，例如，开发人员可以根据页面中的条件动态更改链接文本或目标页。

（2）可以使用数据绑定来指定链接的目标 URL（以及必要时与链接一起传递的参数）。

除了 ID 属性和 Text 属性外，HyperLink 控件还有如下几个常用属性。

（1）ImageUrl：获取或设置该控件显示的图像的路径。

（2）NavigateUrl：获取或设置单击控件时链接到的 URL。

（3）Target：获取或设置单击控件时显示链接到的网页内容的目标窗口或框架。该属性的值包括_blank、_self、_top、_parent 和_search。

使用 HyperLink 控件可单击页面中的文字或图片，打开新的页面，其功能相当于一个链接。如果同时设置了 LinkButton 控件的 ImageUrl 和 Text 属性的值，则 ImageUrl 属性优先。

【范例 1】

结合本节内容，使用 Label 控件显示《早发白帝城》的诗词内容；使用 Literal 控件显示“诗词的作者是”字样；使用文本框供用户输入诗词作者；添加按钮，在单击按钮之后向页面添加 Literal 控件显示诗词的作者；添加 HyperLink 控件指向下一条问题所在的页面，步骤如下。

（1）创建页面并添加 Table 元素，将需要使用的控件依次放在 Table 中的每一行，代码省略。向 Table 中添加 Label 控件显示《早发白帝城》的诗词内容，要求每两句之后添加换行标记，代码如下。

```
<asp:Label ID="Label1" runat="server" Text="Label">
朝辞白帝彩云间，千里江陵一日还。<br/>
两岸猿声啼不住，轻舟已过万重山。<br/>
</asp:Label>
```

（2）向 Table 中添加 Literal 控件显示“诗词的作者是”字样；添加文本框供用户输入诗词作者，代码如下。

```
<asp:Literal ID="Literal1" runat="server">诗词的作者是：</asp:Literal>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

（3）向 Table 中添加 Panel 容器和按钮控件，为按钮控件添加按钮单击事件，代码如下。

```
<asp:Panel ID="Panel1" runat="server">
    <asp:Button  ID="Button1"  runat="server"  Text="  答  案  ：  "
OnClick="Button1_Click1" />
</asp:Panel>
```

（4）向 Table 中添加 HyperLink 控件用于打开下一条问题所在的页面，将新页面的地址设置为“WebForm1.aspx”，代码如下。

```
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl=" WebForm1.
aspx ">下一条</asp:HyperLink>
```

（5）定义按钮的单击事件，向 Panel 容器中添加 Literal 控件显示诗词的作者，代码如下。

```
Literal liter = new Literal();
liter.ID = "literText";
liter.Text = "李白";
Panel1.Controls.Add(liter);
```

（6）运行页面，其效果如图 3-1 所示。单击【答案】按钮，在页面中多了“李白”两个字，其效果如图 3-2 所示。



图 3-1 文本控件显示效果

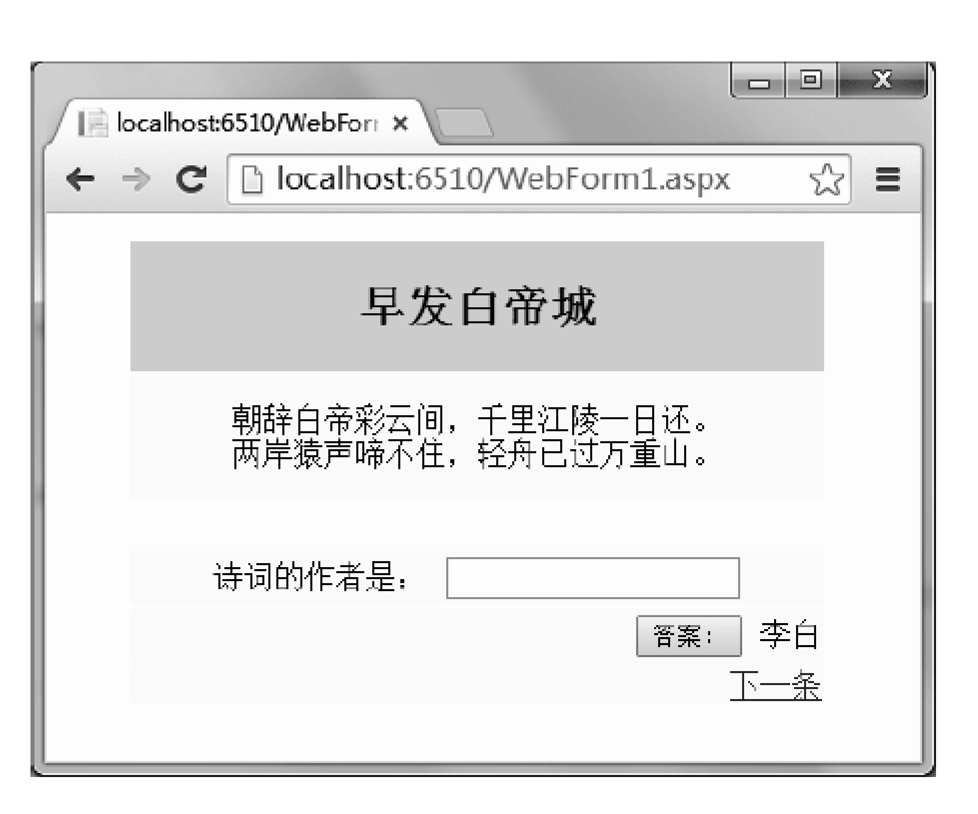


图 3-2 向页面中添加 Literal

3.3 按钮控件

按钮不只是页面中的一个控件，生活中随处可以见到按钮的实例，如手机按键、键盘按键、自动取款机上面的按钮等。按钮通常由用户根据需求向系统发送指令。这个指令是按钮实现定义好的单击（按下）事件，在用户单击按钮时执行。

在 ASP.NET 中除了常见的按钮之外，还提供了可以使用图片背景的按钮，以及可以直接链接到其他页面的链接按钮等，本节将详细介绍。

3.3.1 Button 控件

最为常用的按钮控件是 Button 控件，在 ASP.NET 中 Button 控件在工具箱中的符号为 ab Button。Button 控件常用的属性除了 ID、Text 和 Width 属性外，还有一些自身的属性，如表 3-3 所示。

表 3-3 Button 控件的特有属性

属性名称	说明
CommandArgument	获取或设置可选参数，该参数与关联的 CommandName 一起被传递到 Command 事件
CommandName	获取或设置命令名，该命令名与传递给 Command 事件的 Button 控件相关联
CausesValidation	获取或设置一个值，该值指示在单击控件时是否执行验证
OnClientClick	获取或设置在引发某一个 Button 控件的 Click 事件时所执行的客户端脚本
PostBackUrl	获取或设置单击控件时从当前页发送到的网页的 URL
UseSubmitBehavior	获取或设置一个布尔值，该值指示 Button 控件使用客户端浏览器的提交机制还是 ASP.NET 的回发机制。默认值为 true

表 3-3 中所列举的属性为按钮的功能相关属性，Button 控件最常用的事件是 Click 事件和 Command 事件，它们都在单击 Button 控件时引发，在页面中添加 Button 按钮时

需要指出按钮所关联的事件，而事件所执行的代码需要在页面对应的.cs 文档中进行编写。

在【设计】窗口中双击按钮，会同时在页面和页面对应的.cs 文档中生成按钮的单击事件，默认为 Click 事件。实际上，Command 事件对于按钮并没有什么用，Command 事件的真正作用是作为 BubbleEvent“冒泡”的起点，而 Click 是不冒泡的。在一个含有多个按钮的容器中，按钮的 BubbleEvent 事件捕获到内部的 Command 事件所传递的 CommandName 参数值，以确定与每一个按钮相关联的命令名。

3.3.2 LinkButton 控件

LinkButton 控件又称作链接按钮，其主要功能是执行打开新的页面。LinkButton 控件的外观与 HyperLink 控件相同，但是它实现了与 Button 控件有关的功能。

链接按钮在工具箱中的符号为 LinkButton，它提供了 PostBackUrl 属性，可指向项目中的任意页面或文件。

LinkButton 控件的其他属性、事件以及用法可以参考 Button 控件，这里不再详细介绍。

3.3.3 ImageButton 控件

ImageButton 控件通常被称为图形化按钮，该控件支持使用图片作为按钮的背景，对于页面的美化非常有用。另外，ImageButton 控件还提供了获取图形内单击位置的坐标信息的方法。在工具箱中，该按钮的符号为 ImageButton。

ImageButton 控件的使用与 Button 和 LinkButton 控件大同小异，因此该控件的主要属性和常用事件也可以参考 Button 控件，除了那些属性外，ImageButton 控件还有以下三个常用属性。

（1）ImageUrl：需要在 ImageButton 控件中显示的图像路径。

（2）ImageAlign：获取或设置 Image 控件相对于网页上其他元素的对齐方式。

（3）AlternateText：图像无法显示时显示的文本；如果图像可以显示则表示提示文本。

例如，在 Web 窗体页中添加 ImageButton 控件，为该控件指定其显示的背景图片位置为“~/Styles/first.gif”，若图片无法显示时显示“首页”，按钮代码如下所示。

```
<asp:ImageButton ID="ImageButton1" runat="server" ImageUrl="~/Styles/
first.gif" AlternateText ="首页"/>
```

3.4 选项控件

选项控件又称作选择控件，由开发人员事先将选项列出来，让用户选择。常见的有在填写性别时，提供两个选项（男、女）让用户选择。选项控件将程序能够接受并处理的选项内容列举出来，以防止用户输入的多样性。如同样是性别，用户的输入可能是：男性、男生、男人等词汇，此时程序在对用户信息进行处理时需要对信息进行处理筛选，

增加了程序的复杂度。而使用选项控件可以有效地防止这一情况发生，简化对用户提交信息的处理。

选项控件包括单选框和复选框等，本节将一一介绍。

3.4.1 RadioButton 控件

RadioButton 控件通常被称作单选按钮，其显示样式是一个圆圈，在用户选中时圆圈中心有一个实心圆。RadioButton 控件是单个出现的，通常用于选项较少的选择；若有多个选项可以选择，可以使用 RadioButtonList 控件，该控件是多个 RadioButton 控件的组合控件，显示为一组 RadioButton。

单选按钮在工具箱中的符号为 RadioButton，通过 RadioButton 控件的相关属性可以设置其显示的外观，常用的外观属性如表 3-4 所示。

表 3-4 **RadioButton 控件的常用属性**

属性名称	说明
CausesValidation	获取或设置一个值，该值指示选中控件时是否激发验证。默认值是 false
Checked	控件选中的状态，如果选中该值为 true；否则为 false
GroupName	指定单选按钮所属的组名，在一个组内每次只能选中一个单选按钮
TextAlign	获取或设置与控件关联的文本标签的对齐方式。其值有 Left 和 Right
Text	获取或设置与控件关联的文本标签

单个 RadioButton 控件在用户选中时引发 Changed 事件，默认情况下，该事件并不导致向服务器发送页面，但是通过将 AutoPostBack 属性设置为 true 可以使该控件强制立即发送。用户对单选框的选择结果，通过控件的 Checked 属性来获取。

注 意

若要在选中 RadioButton 控件时将其发送到服务器，浏览器必须支持 ECMAScript（如 JScript 和 JavaScript），并且用户的浏览器要启用脚本撰写。

3.4.2 RadioButtonList 控件

RadioButtonList 控件与 RadioButton 控件所展示的都是一个可选的圆形控件，但 RadioButtonList 控件，常用于多选项的选择。在这个组合中的每一个 RadioButton 控件都是互斥的，即一个 RadioButtonList 中，只能有一个 RadioButton 控件处于选中状态。

RadioButtonList 控件中的选项可以水平排列，也可以垂直排列。其在工具箱中的符号为 RadioButtonList，默认为垂直排列。

由于 RadioButtonList 控件派生自 ListControl 基类，因此工作方式与列表控件（如 ListBox、DropDownList 和 BulletedList 等）相似。

RadioButtonList 控件包含 RadioButton 控件的多个属性，同时有着自己独特的属性。RadioButtonList 还可以动态地绑定数据源，其常用属性如表 3-5 所示。

表 3-5　RadioButtonList 控件的常用属性

属性名称	说明
DataSourceID	获取或设置控件的 ID，数据绑定控件从该控件中检索其数据项列表
DataSource	指定该控件绑定的数据源
DataTextField	获取或设置为列表项提供文本内容的数据源字段
DataValueField	获取或设置为各列表项提供值的数据源字段
Item	列表控件项的集合
SelectedIndex	获取或设置列表中选中项的最低序号索引
SelectedItem	获取列表控件中索引最小的选定项
SelectedValue	获取列表控件中选定项的值，或选择列表控件中包含指定值的项
RepeatColumns	获取或设置在该控件上显示的列数
RepeatDirection	获取或设置组中单选按钮的显示方向，它的值有 Vertical（默认值）和 Horizontal
RepeatLayout	获取或设置一个值，该值指定是否使用 table 元素、ul 元素、ol 元素或 span 元素呈现列表。其值分别是 Table、Flow、UnorderedList 和 OrderedList

与单个 RadioButton 控件相反，RadioButtonList 控件在用户更改列表中选定的单选按钮时会引发 SelectedIndexChanged 事件。默认情况下，该事件并不导致向服务器发送页面，但是可以通过将 AutoPostBack 属性设置为 true 来指定此选项。

单选按钮很少单独使用，对单选按钮分组时有两种方式，如下所示。

（1）先向页中添加单个 RadioButton 控件，然后将所有这些控件手动分配到一个组中。

（2）向页中添加 RadioButtonList 控件，该控件中的列表将自动分组。

3.4.3　CheckBox 控件

CheckBox 控件也叫复选框，它在 Web 窗体页面上显示为一个可选的方块，常用于为用户提供多项选择。

CheckBox 控件与 RadioButton 控件功能类似，不同的是，RadioButton 控件显示为圆圈可选框，而 CheckBox 控件显示为方形可选框，其在工具箱中的符号为☑ CheckBox。CheckBox 控件的一般形式如下。

```
<asp:CheckBox ID="控件名称" runat="server" AutoPostBack="true | false"
Checked="true | false" Text="控件文字" TextAlign="left | right"
OnCheckedChanged="事件程序名称" />
```

从语法形式中可以看出，除了 ID 和 runat 属性外，CheckBox 控件最常用的有 4 个属性和一个事件。属性的说明如下。

（1）AutoPostBack：默认值为 false，设置当使用者选择不同的项目时，是否自动触发 CheckedChanged 事件。

（2）Checked：该属性传回或设置是否该项目被选取。

（3）TextAlign：该属性设置控件所显示的文字是在按钮的左方还是右方。

（4）Text：该属性设置 CheckBox 控件所显示的文本内容。

单个 CheckBox 控件在用户单击该控件时会引发 CheckedChanged 事件，但是由于

AutoPostBack 属性的值为 false，因此在默认情况下，该事件并不导致向服务器发送页面。通过 Checked 属性，可获取该控件是否为选中状态。

3.4.4 CheckBoxList 控件

与 RadioButtonList 控件和 RadioButton 控件之间的关系一样，CheckBox 控件也有着用于多项选择的 CheckBoxList 控件，其在工具箱中的符号为 CheckBoxList。

CheckBoxList 控件与 RadioButtonList 不同，RadioButtonList 中只能有一个选项被选中，而 CheckBoxList 支持同时选中多个选项。如网购大衣的人对大衣的材质要求不高，可以是棉的也可以是羊毛的或羊绒的，那么该客户在选择大衣的时候可同时在材质一栏选择棉、羊毛和羊绒这三个选项。

如果想用数据库中的数据创建一组复选框，那么 CheckBoxList 控件是很好的选择。CheckBoxList 控件也可以动态地绑定数据源，其常用属性如表 3-6 所示。

表 3-6 **CheckBoxList 控件的常用属性**

属性名称	说明
DataSourceID	获取或设置控件的 ID，数据绑定控件从该控件中检索其数据项列表
DataSource	指定该控件绑定的数据源
DataMember	用户绑定的表或视图
DataTextField	获取或设置为列表项提供文本内容的数据源字段
DataTextFormatString	获取或设置格式化字符串，该字符串用来控制如何显示绑定到列表控件的数据
DataValueField	获取或设置为各列表项提供值的数据源字段
Items	获取列表项的集合
SelectedIndex	获取或设置列表中选中项的最低序号索引
SelectedItem	获取列表控件中索引最小的选定项
SelectedValue	获取列表控件中选定项的值，或选择列表控件中包含指定值的项
RepeatColumns	获取或设置在该控件上显示的列数
RepeatDirection	获取或设置组中单选按钮的显示方向，它的值有 Vertical（默认值）和 Horizontal
RepeatLayout	获取或设置一个值，该值指定是否使用 table 元素、ul 元素、ol 元素或 span 元素呈现列表

CheckBoxList 控件与 CheckBox 控件在事件处理方面也有不同，CheckBoxList 控件在选项改变时会引发 SelectedIndexChanged 事件。默认情况下，该事件并不导致向服务器发送窗体，但是可以通过 AutoPostBack 属性的值来控制。

由于 CheckBoxList 控件可同时选择多个选项，因此不能像 RadioButtonList 控件一样获取选中值，而是需要遍历其所有选项，以获取哪些选项被选中。

【范例 2】

结合本节内容，创建学生参加校园活动的统计表，要求学生填写姓名，选择性别和所参加的校园活动。

由于学生的性别是男和女这两个互斥选项，因此可以使用 RadioButtonList 控件；由

于一名学生可以同时参与多个校园活动，因此可以使用 CheckBoxList 控件，步骤如下。

（1）省略页面的部分代码，页面中需要有填写学生姓名的文本框、有供学生选择性别的单选框和供学生选择参与活动的复选框，还有【提交】按钮和信息显示标签，这些控件的页面代码如下。

```
<asp:TextBox ID="nameBox" runat="server"></asp:TextBox></td>
<asp:RadioButtonList ID="Sex" runat="server" RepeatDirection=
"Horizontal" Width="120px">
    <asp:ListItem Selected="True">男</asp:ListItem>
    <asp:ListItem>女</asp:ListItem>
</asp:RadioButtonList>
<asp:CheckBoxList ID="sleeveA" runat="server" RepeatDirection=
"Horizontal"
    Width="492px">
    <asp:ListItem Selected="True">体育项目</asp:ListItem>
    <asp:ListItem>舞蹈</asp:ListItem>
    <asp:ListItem>武术</asp:ListItem>
    <asp:ListItem>音乐</asp:ListItem>
    <asp:ListItem>计算机软件</asp:ListItem>
    <asp:ListItem>计算机硬件</asp:ListItem>
</asp:CheckBoxList>
<asp:Button ID="Submit" runat="server" Text="提交" BackColor="#FFFFCC"
BorderColor="#FFD800" BorderStyle="Outset" Height="24px" Width="103px"
OnClick="Submit Click" /></td>
<asp:Label ID="message" runat="server" Text=""></asp:Label></td>
```

（2）为【提交】按钮添加鼠标单击事件，获取学生的信息并显示在页面上，代码如下。

```
protected void Submit Click(object sender, EventArgs e)
{
    if (nameBox.Text == "")
    {
        message.Text = "姓名不能为空";
        message.ForeColor = System.Drawing.Color.Red;
    }
    else
    {
        string sname = nameBox.Text;
        string ssex = Sex.SelectedValue;
        string active = "";
        foreach (ListItem item in sleeveA.Items)     //遍历复选框
        {
            if (item.Selected)                       //判断选项是否被选中
            { active += item.Value + "、"; }
        }
        message.Text = "姓名：" + sname+ " 性别：" + ssex + "<br/>"+"爱好：" 
        +active;
```

```
        }
    }
```

（3）运行该页面，在没有任何输入的情况下单击【提交】按钮，其效果如图 3-3 所示。由于用户没有输入姓名，因此有红色字体的提示语句，没有读取用户的输入信息。



图 3-3　单选框和复选框的应用

（4）向页面中输入数据并单击【提交】按钮，其效果如图 3-4 所示。在提交之后获取了用户的输入信息，并成功赋值给名为 message 的标签控件来显示。由于图 3-4 是在图 3-3 的基础上重新提交，因此字体显示为红色；若是直接在填写完整的情况下提交的，字体显示为黑色。



图 3-4　单选框和复选框的数据获取

3.5 列表控件

除了选择控件以外，还有一些列表控件同样可以提供选项的选择。不同的是，列表控件在选项的控制上更为灵活；而且无论选项数量有多大，其在页面中所占有的空间是固定的。本节详细介绍 ASP.NET 中的列表控件。

3.5.1 DropDownList 控件

DropDownList 控件又称作下拉框控件，能够让用户在下拉列表中进行选择，该控件在工具箱中的符号为 DropDownList。

开发人员也可以把 DropDownList 控件看作容器，这些列表项都属于 ListItem 类型，每一个 ListItem 对象都是带有单独属性（如 Text 属性、Selected 属性和 Value 属性）的对象。DropDownList 控件只能选择一个选项。

DropDownList 控件在网页中，较为常用的是对地区的选择，即提供【省份】下拉框和【市区】下拉框，在选择省份之后，【市区】下拉框将获取该省所管辖的市区，供用户选择。

开发人员可以通过 DropDownList 控件的 Width 和 Height 来控制其外观，部分浏览器不支持以像素为单位设置的高度和宽度时，这些浏览器将使用行计数设置。除了 Width 和 Height 外，表 3-7 列举了其他的常用属性。

表 3-7 DropDownList 控件的常用属性

属性名称	说明
AutoPostBack	获取或设置一个值，该值指示用户更改列表中的内容时是否自动向服务器回发
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DataTextField	获取或设置为列表项提供文本内容的数据源字段
DataValueField	获取或设置各列表项提供值的数据源字段
Items	获取列表控件项的集合
SelectedIndex	获取或设置列表中选定项的最低索引
SelectedItem	获取列表控件中索引最小的选定项
SelectedValue	获取列表控件中选定项的值，或选择列表控件中包含指定值的项

除了属性外，DropDownList 控件经常使用 SelectedIndexChanged 事件，当用户修改当前项时将会引发该事件。

3.5.2 BulletedList 控件

BulletedList 控件所显示的是一个列表，如新闻页面中列举的前 10 条新闻标题，BulletedList 控件在列举新闻标题的同时，还可以为每一条标题提供一个链接，通往该新闻的详细信息页面。

BulletedList 控件在工具箱中的符号为 BulletedList，可以在页面中创建一个无序或有序（编号的）的项列表。BulletedList 控件所呈现的样式相当于 HTML 中的 ul 或 ol 元素。通过 BulletedList 控件可以实现以下效果。

（1）可以指定项、项目符号或编号的外观。

（2）静态定义列表项或通过将控件绑定到数据来定义列表项。

（3）也可以在用户单击项时做出响应。

通过创建静态项或将控件绑定到数据源，可以定义 BulletedList 控件的列表项。而且

通过该控件的相关属性可以设置其外观效果，如表 3-8 所示为 BulletedList 控件的常用属性，并对这些属性进行了说明。

表 3-8　BulletedList 控件的常用属性

属性名称	说明
AppendDataBoundItems	获取或设置一个值，该值指示是否在绑定数据之前清除列表项。默认值为 false
BulletImageUrl	获取或设置为控件中的每个项目符号显示的图像路径，把 BulletStyle 的值设置为 CustomImage 时有效
BulletStyle	获取或设置控件的项目符号样式
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DataTextField	获取或设置为列表项提供文本内容的数据源字段
DataValueField	获取或设置为列表项提供值的数据源字段
DisplayMode	获取或设置控件中的列表内容的显示模式。其值包括 Text（默认值）、HyperLink 和 LinkButton
FirstBulletMember	获取或设置排序控件中开始列表项编号的值
Items	获取列表控件项的集合

BulletedList 控件可以通过 BulletStyle 属性自定义列表项外观，如果将控件设置为呈现项目符号，则可以选择与 HTML 标准项目符号样式匹配的预定义项目符号样式字段。BulletStyle 属性的可取值有 10 个，同一个值不同的浏览器所呈现项目符号的方式会不同，甚至有些浏览器不支持特定的项目符号样式（如 Disc 字段）。BulletStyle 属性的值如表 3-9 所示。

表 3-9　BulletStyle 属性可取值及其说明

可取值	说明
NotSet	未设置
Numbered	数字
LowerAlpha	小写字母
UpperAlpha	大写字母
LowerRoman	小写罗马数字
UpperRoman	大写罗马数字
Disc	实心圆
Circle	圆圈
Square	实心正方形
CustomImage	自定义图像

通过 BulletedList 控件添加集合项时有多种方法，例如，设计窗体添加 BulletedList 控件，该控件右端有箭头按钮，单击按钮有两条可选的链接，一个是【选择数据源】，一个是【编辑项】。选择【编辑项】，可打开【ListItem 集合编辑器】，如图 3-5 所示。

在如图 3-5 所示的对话框中单击【添加】按钮，可添加新的集合项并打开对应集合项的属性列表，如图 3-6 所示。

在属性列表中分别设置 Text 属性和 Value 属性。Text 属性定义控件在页上显示的内容；Value 属性定义第二个值，该值不会显示，但用户在选择某个项时能返回该值。

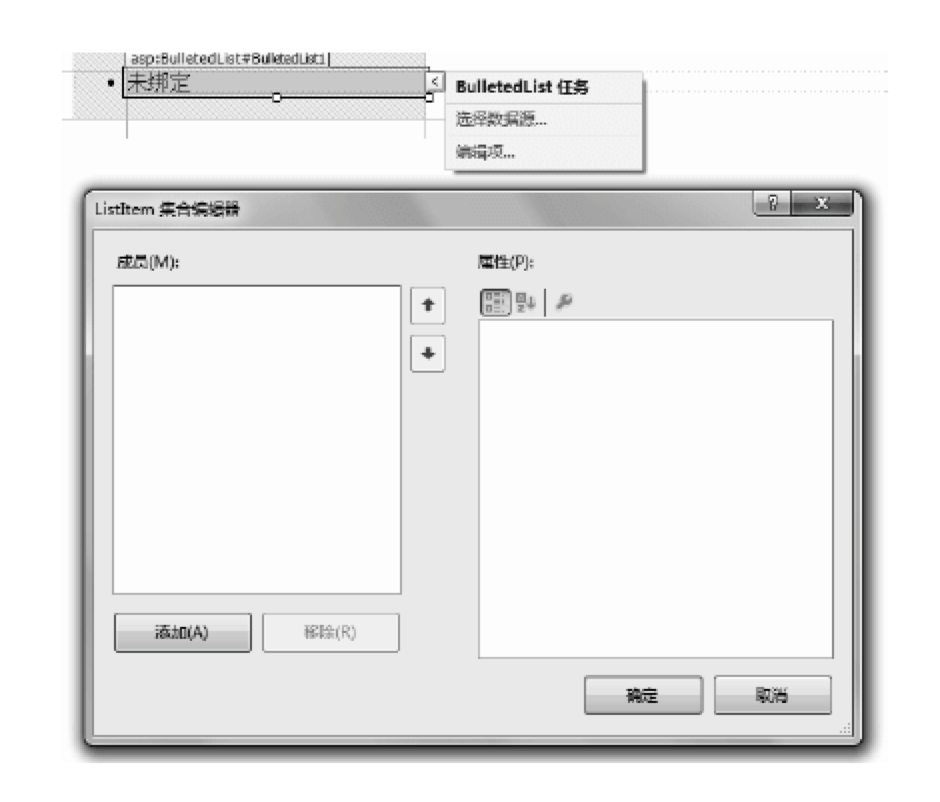


图 3-5　集合编辑器

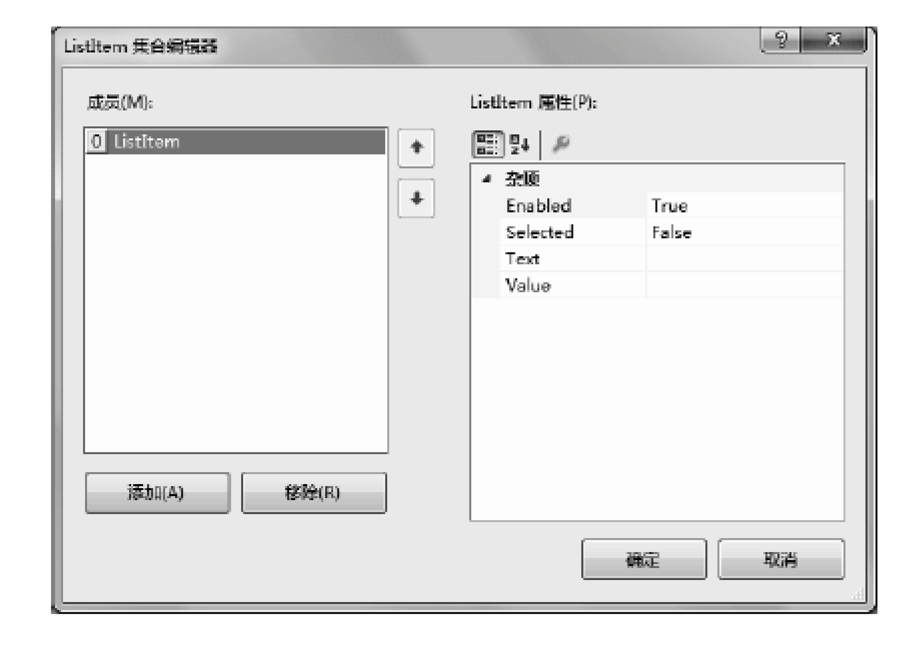


图 3-6　集合项属性

所有项添加完成后单击图 3-6 中的【确定】按钮进行添加，添加完成后，页面上将有如下新增代码。

```
<asp:BulletedList ID="BulletedList1" runat="server" BulletStyle=
"Circle">
    <asp:ListItem>新建项 1</asp:ListItem>
    <asp:ListItem>新建项 2</asp:ListItem>
    <%--省略其他相关 ListItem 代码--%>
</asp:BulletedList>
```


BulletedList 控件的 Items 属性返回所有集合项的列表对象，该对象包含多个方法，通过这些方法可以添加指定的项、删除项或查找项等，它们的具体说明如表 3-10 所示。

表 3-10　集合列表项的常用方法

方法名称	说明
Add()	将表示指定字符串的 ListItem 项添加到集合的结尾
AddRange()	将 ListItem 数组中的项添加到集合
Clear()	从集合中移除所有的 ListItem 项
CopyTo()	将集合中的项复制到 Array 中，从指定的索引开始复制
Insert()	将指定的 ListItem 插入到集合中的指定索引位置
Remove()	从集合中移除指定的 ListItem
RemoveAt()	从集合中移除指定索引位置的 ListItem

集合列表项除了上面列举的方法外，还有一个最常用的属性 Count，该属性获取集合中 ListItem 的对象总数。例如，开发人员分别通过 Add()、AddRange()方法和 Insert()方法向集合中添加项目，然后使用 RemoveAt()进行删除，代码如下。

```
BulletedList1.Items.Add(new ListItem("Add()方法添加内容", "Add()方法添加内
容"));
ListItem[] li = { new ListItem("AddRange()添加 1", "AddRange()添加 1"), new
ListItem("AddRange()添加 2", "AddRange()添加 2") };
BulletedList1.Items.AddRange(li);
BulletedList1.Items.Insert(0, new ListItem("插入到第1位", "插入到第一位"));
BulletedList1.Items.RemoveAt(1);
```

3.5.3 ListBox 控件

ListBox 控件是另一种形式的列表控件，其列举项的添加和删除通常仅通过后台代码来控制。

ListBox 控件允许用户从预定义的列表中选择一项或多项，其显示的列表项目数量根据控件的大小来决定。其在工具箱中的符号为 ListBox。读者可以从以下两个方面控制列表的外观。

（1）显示的行数：可以将该控件设置为显示特定的项数，如果该控件包含比设置的项数更多的项，则显示一个垂直滚动条。

（2）宽度和高度：可以以像素为单位设置控件的大小。在这种情况下，控件将忽略已设置的行数，而是显示足够多的行直至填满控件的高度。

1. ListBox 控件的属性

ListBox 控件有许多常用属性，如 DataSource、AutoPostBack、SelectedIndex 和 SelectedValue 等，表 3-11 对常用的属性进行了说明。

表 3-11 **ListBox 控件的常用属性**

属性名称	说明
AutoPostBack	获取或设置一个值，该值指示当用户更改列表中的选定内容时是否自动向服务器回发
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DataTextField	获取或设置为列表项提供文本内容的数据源字段
DataValueField	获取或设置各列表项提供值的数据源字段
Items	获取列表控件项的集合
Rows	获取或设置该控件中显示的行数
SelectedIndex	获取或设置列表中选定项的最低索引
SelectedItem	获取列表控件中索引最小的选定项
SelectedValue	获取列表控件中选定项的值，或选择列表控件中包含指定值的项
SelectionMode	获取或设置控件的选择模式，它的值有两个，分别为 Single 和 Multiple。默认为 Single

通常情况下，用户可以通过单击列表中的单个项来选择它。如果将 ListBox 控件 SelectionMode 属性的值设置 Multiple（即允许进行多重选择），则用户可以在按住 Ctrl 或 Shift 键的同时，单击以选择多个项。

如下代码通过 SelectedIndex 属性设置了 ListBox 控件的索引项。

```
ListBox1.SelectedIndex = 2;
```

2. ListBox 控件的事件

当用户选择某一项时，ListBox 控件就会引发 SelectedIndexChanged 事件。但是默认情况下，该事件不会导致将页发送到服务器。例如，在 ListBox 的 SelectedIndexChanged

事件中判断某一项是否选中，代码如下。

```
protected void ListBox1 SelectedIndexChanged(object sender, EventArgs e)
{
    foreach (ListItem item in ListBox1.Items){
        if (item.Selected) {
            //省略代码
        }
    }
}
```

3. 向 ListBox 控件添加选项

ASP.NET 中可以使用三种方式向 ListBox 控件中添加项，如下所示。

（1）在设计时添加静态项。

（2）使用编程的方式在运行时添加项。

（3）使用数据绑定添加项。

如下代码通过 DataSource、DataTextField 和 DataValueField 属性演示了如何通过编程动态绑定数据。

```
ListBoxShow.DataSource = GetPlaceList();    //该方法可以返回 DataTable 对象
ListBoxShow.DataTextField = "placeName";    //对应后台数据库中的字段名
ListBoxShow.DataValueField = "placeId";     //对应后台数据库的字段名
```

4. 确定 ListBox 控件的所选内容

使用 ListBox 控件时最常见的内容是确定用户已选择了哪一项或哪些项，其主要取决于该控件允许单项选择还是多重选择。有两种方式确定单项选择列表控件的选定内容，如下所示。

（1）如果获取选择项的索引值，需要使用 SelectedIndex 属性的值。该属性的索引是从 0 开始的，如果没有选择任何项，则该属性的值是–1。

（2）如果获取选择项的内容，需要使用 SelectedItem 属性，该属性返回一个 ListItem 类型的对象。通过该对象的 Text 属性或 Value 属性可以获取选择项的内容。

如果 ListBox 控件允许多重选择时，确定 ListBox 控件所选中的内容，需要依次通过控件的 Items 集合，分别测试每一项的 Selected 属性。主要代码如下。

```
Protected void Button1 Click(object sender, System.EventArgs e)
{
    string msg = "" ;
    foreach(ListItem li in ListBox1.Items)          //遍历集合中的内容
    {
        if(li.Selected == true)                     //判断某一项是否选中
          {
            msg += "<br>" + li.Text + " is selected.";
          }
    }
    Label1.Text = msg;
}
```

3.6 容器控件

容器控件在页面中的作用相当于一个容器，容器控件是没有显示样式的，但容器控件可将其他控件（如前面几节介绍的控件）包含在内，形成一个分组，并对其内部的控件单独执行。

容器控件可以作为 Web 服务器控件、HTML 服务器控件和 HTML 元素对象的父控件，ASP.NET 中提供了多个与容器相关的控件，本节将介绍最常用的两个控件：MultiView 控件和 Panel 控件。

3.6.1 Panel 控件

Panel 控件是容器控件的一种，可以容纳其他控件，并将其子控件组合为一个整体。Panel 控件通常会被称作面板。Panel 控件在工具箱中的符号为 Panel。

Panel 控件在 3.2.2 节中使用过，即向 Panel 控件中添加 Literal 控件，Literal 控件是可以通过代码添加在容器中的。

当开发人员需要以编程的方式创建内容并需要一种将内容插入到页中的方法时，Panel 控件最为适用。该控件的常用三种方式如下。

（1）容纳其他控件，将其他控件组合为一个整体进行管理。

（2）可以在 Panel 内设置默认按钮，当用户在 Panel 面板的文本框输入时按 Enter 键，这与用户单击特定的默认按钮具有相同的效果。

（3）部分控件（如 TreeView）没有内置的滚动条，通过在 Panel 控件中放置滚动条控件可以添加滚动行为。同时可以使用 Panel 控件的 Height 和 Width 属性添加滚动条，将 Panel 控件设置为特定的大小，然后再设置 ScrollBars 属性。

Panel 控件有多个属性分别设置控件的外观和文本方向等数据，如表 3-12 所示为 Panel 控件的常用属性。

表 3-12 **Panel 控件的常用属性**

属性名称	说明
BackImageUrl	控件背景图像文件和 URL
DefaultButton	控件中默认按钮的 ID
Direction	控件中内容的显示方向，默认值为 NoSet，其他的值有 LeftToRight 和 RightToLeft
GroupingText	获取或设置控件中包含的控件组的标题，如果指定了滚动条则设置该属性将不显示滚动条
ScrollBars	获取或设置控件中滚动条的可见性和位置
HorizontalAlign	获取或设置面板内容的水平对齐方式
Wrap	获取或设置一个指示面板中内容是否换行的值

注 意

不能够在 Panel 控件中同时指定滚动条和分组文本，如果设置了分组文本，其优先级高于滚动条。

3.6.2 MultiView 控件

MultiView 控件通常用作 View 控件组的容器。它允许开发人员定义一组有着子控件的 View 控件。

MultiView 和 View 控件用作其他控件的容器，并提供一种视图切换方式。MultiView 和 View 控件结合，可制作出选项卡的效果。MultiView 在工具箱中的符号为 MultiView，可执行以下两个任务。

（1）根据用户选择或其他条件提供备选控件集。

（2）创建多页窗体。

由于MultiView控件通常和View控件一起使用，因此该控件通常被称作多视图控件。一个 MultiView 中可以包含多个 View 控件，但是 MultiView 控件的当前活动控件只能是这些 View 控件中的一个。无论是 MultiView 控件还是各个 View 控件，除了当前 View 控件的内容外，都不会在页面中显示任何标记。

MultiView 控件最常用的属性有两个：ActiveViewIndex 和 Views。ActiveViewIndex 用来获取或设置 MultiView 控件的活动 View 控件的索引；而 Views 控件用来获取 MultiView 控件的 View 控件的集合。

设置 MultiView 控件的 ActiveViewIndex 属性可以在视图间移动。另外，MultiView 控件还支持可以添加到每个 View 控件的导航按钮。如果要创建导航按钮，可以向每个 View 控件添加一个按钮控件（Button、LinkButton 或 ImageButton）。然后可以将每个按钮的 CommandName 和 CommandArgument 属性设置为保留值，这些保留值以使 MultiView 控件移动到另一个视图。如表 3-13 所示列出了 CommandName 值和相对应的 CommandArgument 值。

表 3-13 **CommandName 和 CommandArgument 值**

CommandName 值	CommandArgument 值
NextView	（没有值）
PrevView	（没有值）
SwitchViewByID	要切换到的 View 控件的 ID
SwitchViewByIndex	要切换到的 View 控件的索引号

3.7 其他控件

除了上述控件以外，还有其他常用控件，如图片显示控件、日历控件和广告控件等。

几乎每一个网页中都有图片的显示，图片显示使用 Image 控件，而根据图片的不同区域，响应不同事件则需要使用 ImageMap 控件。此外，还有常见的时间控件，可供用户选择或查阅日期，本节将详细介绍。

3.7.1 图片显示控件

图片显示控件是一个容易操作的控件，其图片的显示正如标签控件显示文本一样。

通过为图片控件指定图片位置，即可显示指定图片。

图片显示控件为 Image 控件，其在工具箱中的符号为 Image。与其他控件不同，Image 控件不支持任何事件，通常情况下，将通过使用 ImageMap 或 ImageButton 控件来创建交互式图像。

但 Image 控件有着常用属性，可在代码中通过属性来控制 Image 控件对图片的显示，包括图片的位置、大小等，常见的属性如表 3-14 所示。

表 3-14 Image 控件的常用属性

属性名称	说明
Width	显示图像的宽度
Height	显示图像的高度
ImageAlign	图像的对齐方式
ImageUrl	要显示图像的 URL

除了显示图像外，Image 控件还可以为图像指定各种类型的文本。与之相关的属性如下所示。

（1）ToolTip：在一些浏览器中作为工具提供显示的文本。

（2）AlternateText：在无法找到图形文件时显示的文本。如果没有指定任何 ToolTip 属性，某些浏览器将使用 AlternateText 值作为工具提示。

（3）GenerateEmptyAlternateText：该属性的值如果为 true，则所呈现的图像元素的 alt 特性将设置为空字符串。

ImageUrl 属性可以显示指定的文本，还可以通过该属性绑定数据源，以显示数据库后台的图像。Image 控件的简单使用代码如下。

```
<asp:Image ID="imgShow" ImageUrl="~/sg icon.png" runat="server" Width=
"200px" Height="200px" />
```

上述代码为图片显示控件指定了图片位置“~/sg_icon.png”，以及图片显示的宽度 200px 和高度 200px。

3.7.2 图片响应控件

ImageMap 控件同样可以显示图像，但该控件可提供三种不同区域，以响应不同的事件。ImageMap 控件在工具箱中的符号为 ImageMap，若添加了该控件，其区域内部可添加三种区域，代码如下。

```
<asp:ImageMap ID="ImageMap1" runat="server" ImageUrl="~/Styles/
1045.jpg">
<asp:CircleHotSpot />
<asp:PolygonHotSpot />
<asp:RectangleHotSpot />
</asp:ImageMap>
```

ImageMap 控件创建具有用户可以单击的单个区域的图像，这些单个的区域点称为

作用点。每一个作用点都可以是一个单独的超链接或回发事件。

ImageMap 控件由两个元素组件：一个是图像，它可以是任何标准的 Web 图形格式的图形（如.jpg、.gif 或.png）；另一个是 HotSpot 的集合，每个作用点都是一个类型为 CircleHotSpot、RectangleHotSpot 和 PolygonHotSpot 的不同项。

ImageMap 控件可以包含 Image 控件的属性，除了那些属性外，该控件还有两个重要属性：HotSpots 和 HotSpotMode。

HotSpotMode 属性表示获取或设置单击 HotSpot 对象时该控件对象的默认行为。它的值是 HotSpotMode 枚举的值之一，具体说明如下。

（1）NotSet：未设置。默认情况下控件会执行导航操作，即导航到指定的网页；如果未指定导航的网页，则导航到当前网站的根目录。

（2）Navigate：导航到指定的网页，如果未指定导航的网页则导航到当前网站的根目录。

（3）PostBack：执行回发操作，用户单击区域时执行预先定义的事件。

（4）Inactive：无任何操作，这时该控件和 Image 控件的效果一样。

ImageMap 控件在页面中的使用，首先需要为该控件添加相应的区域，接着是对其单击事件代码的编写。如分别定义三种区域代码如下。

```
<asp:ImageMap ID="ImageMap1" ImageUrl="~/Styles/1045.jpg " runat=
"server" ImageAlign="Left" style="border:none" onclick="ImageMap1_
Click" Width="500">
    <asp:CircleHotSpot Radius="80" X="100" Y="100" AlternateText="圆形区
    域" HotSpotMode="PostBack" PostBackValue="CH" />
    <asp:RectangleHotSpot Bottom="200" Left="300" Right="500" Top="0"
    HotSpotMode="PostBack" AlternateText="方形区域" PostBackValue="RH" />
    <asp:PolygonHotSpot Coordinates="100,100,300,300,200,300" HotSpotMode=
    "PostBack" PostBackValue="PH" AlternateText="多边形区域" />
</asp:ImageMap>
```

接下来可定义其单击事件，如单击不同区域，弹出不同内容的对话框，其单击事件代码如下所示。

```
protected void ImageMap1 Click(object sender, ImageMapEventArgs e)
{
   String region = "";                    //弹出对话框的显示文字
   switch (e.PostBackValue)
   {
      case "CH":
         region = "圆形区域";
         break;
      case "RH":
         region = "方形区域";
         break;
      case "PH":
         region = "多边形区域";
         break;
```

```
        }
        Page.ClientScript.RegisterStartupScript(GetType(), "", "<script>alert
        ('"+region+"');</script>");//弹出对话框
    }
```

3.7.3 日历控件

一些网站中会有时间的选择或显示，可以使用日历控件 Calendar 控件。但对于日期的选择或显示，目前有多种第三方的控件可以使用，本节介绍 ASP.NET 中自带的控件。

Calendar 控件在工具箱中的符号为 Calendar，它显示日历中的可选日期，并显示与特定日期关联的数据。使用该控件可以执行以下操作。

（1）捕获用户交互（例如在用户选择一个日期或一个日期范围时）。

（2）自定义日历的外观。

（3）在日历中显示数据库中的信息。

Calendar 控件可以显示某个月的日历，也允许用户选择日期，还可以跳转日期到前一个或下一个月，如表 3-15 所示对 Calendar 控件的主要属性进行了具体说明。

表 3-15 Calendar 控件的主要属性

属性名称	说明
Caption	日历的标题
CaptionAlign	日历标题文本的对齐方式。其值包含 NotSet（默认值）、Bottom、Left、Right 和 Top
DayNameFormat	获取或设置周中各天的名称格式。其值包括 Short（默认值）、Full、FirstTwoLetters、FirstLetter 和 Shortest
FirstDayOfWeek	获取或设置要在控件的第一天列中显示的一周中的某天。默认值为 Default
NextMonthText	获取或设置为下一个月导航控件显示的文本
NextPrevFormat	获取或设置控件的标题部分中下个月和上个月导航元素的格式。其值包括 CustomText（默认值）、FullMonth 和 ShortMonth
PrevMonthText	获取或设置为前一个月导航控件显示的文本
SelectedDate	获取或设置选定的日期
SelectionMode	获取或设置日期的选定模式。其值包括 Day（默认值）、DayWeek、DayWeekMonth 和 None
SelectMonthText	获取或设置为选择器列中月份选择元素显示的文本
SelectWeekText	获取或设置为选择器列中周选择元素显示的文本
ShowDayHeader	获取或设置一个值，该值指示是否显示一周中各天的标头。默认值为 true
ShowGridLines	获取或设置一个值，该值指示是否用网格线分隔控件上的日期。默认值为 false
ShowNextPrevMonth	获取或设置一个值，该值指示控件是否在标题部分显示下个月和上个月导航元素。默认值为 false
TitleFormat	获取或设置标题部分的格式。默认值为 MonthYear

读者在调用 SelectedDate 属性获取选定的日期时，该属性返回一个 DateTime 对象，通过 DateTime 对象的各个方法可以获取不同的内容。例如，调用 Add()方法可以返回一个新的日期对象；调用 ToShortDateString()方法将 DateTime 对象的值转换为其等效的短

日期字符串表示形式。

> **提示**
> **除了上述属性外，Calendar 控件还可以使用其他属性指定外观内容。如 ForeColor 属性设置文本的颜色；DayHeaderStyle 属性设置标头行的样式；以及 BackColor 属性设置背景颜色等。**

Calendar 控件主要使用两个事件：DayRender 和 SelectionChanged。当控件创建要发送到浏览器的输出时引发 DayRender 事件，在准备要显示的日时将为每个日引发该事件；当用户通过单击日期选择器控件选择一天、一周或整月时引发该事件。

在 DayRender 事件的方法中带有两个参数：包括对引发事件的 Calendar 控件的引用和 DayRenderEventArgs 类型的对象。该对象提供对另外两个对象的访问。

（1）Cell：它是一个 TableCell 对象，可以用于设置个别日的外观。

（2）Day：可以用于查询关于呈现日的信息。该对象不仅支持各种可用于了解有关日的信息的属性（如 IsSelected 和 IsToday 等），还支持 Controls 集合，可操作该集合以将内容添加到日中。

3.7.4 广告控件

广告控件 AdRotator 控件用来在页面上生成随机广告性质的元素，并通过 AdvertisementFile 属性获取或设置包含广告信息的 XML 文件的路径。

AdRotator 控件通常获取 XML 文件中的信息，并将广告信息显示出来。其在工具箱中的符号为 AdRotator，只需创建好相关的 XML 文件，并通过 AdvertisementFile 属性获取 XML 文件的路径即可，在 XML 文件中，需要对广告进行设置的属性如表 3-16 所示。

表 3-16 广告信息属性

属性说明	描述
ImageUrl	图像文件的绝对路径或相对地址
NavigateUrl	当图像被单击时，可访问相应的网页
AlternateText	当鼠标移动到图片上方时，将显示的提示信息
keyword	指定广告的分类，可以利用此属性来对广告条进行分类
Impressions	指定图片显示频率

AdRotator 控件的 AdvertisementFile 属性获取 XML 文件，必须确保 XML 文件内容的规范性。如创建一个 XML 文件，则其文件内容格式如下。

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
  <Ad>
    <ImageUrl>Styles/1045.jpg</ImageUrl>
    <NavigateUrl>广告链接地址</NavigateUrl>
    <AlternateText>提示信息 </AlternateText>
    <Keyword>广告分类/关键字</Keyword>
```

```
    <Impressions>该广告出现的频率</Impressions>
  </Ad>
  <Ad>
    <ImageUrl>显示图片的地址</ImageUrl>
    <NavigateUrl></NavigateUrl>
    <AlternateText></AlternateText>
    <Keyword> </Keyword>
    <Impressions>10</Impressions>
  </Ad>
</Advertisements>
```

3.8 实验指导——常识调查页面

结合本章内容，创建常识调查页面，测试小学生对常识的了解情况。要求分别添加一道单选题和一道多选题，在提交之后显示学生的答案和正确答案；在提交之前需要学生选择自己所在的年级，步骤如下。

（1）首先是页面的创建，步骤省略。学生需要选择自己所在的年级，可使用下拉列表；选择题的选项是需要列举出来让学生选择的，因此可分别使用 RadioButtonList 控件和 CheckBoxList 控件来编写单选题和多选题。另外，需要【提交】按钮和用来显示信息的标签，并额外添加一个容器，显示正确答案。上述过程所涉及的控件代码如下。


```
<asp:DropDownList ID="classList" runat="server" Height="20px" Width=
"80px">
    <asp:ListItem Selected="True">一年级</asp:ListItem>
    <asp:ListItem>二年级</asp:ListItem>
    <asp:ListItem>三年级</asp:ListItem>
    <asp:ListItem>四年级</asp:ListItem>
</asp:DropDownList>
  <asp:RadioButtonList ID="direction" runat="server">
      <asp:ListItem>东</asp:ListItem>
      <asp:ListItem>西</asp:ListItem>
      <asp:ListItem>南</asp:ListItem>
      <asp:ListItem>北</asp:ListItem>
  </asp:RadioButtonList>
<asp:CheckBoxList ID="province" runat="server">
    <asp:ListItem>哈尔滨</asp:ListItem>
    <asp:ListItem>黑龙江</asp:ListItem>
    <asp:ListItem>吉林</asp:ListItem>
    <asp:ListItem>辽宁</asp:ListItem>
</asp:CheckBoxList>
<asp:Button ID="Submit" runat="server" Text="提交" BackColor="#E2F2FF"
BorderColor="#007ACC" BorderStyle="Outset" Height="24px" Width="103px"
OnClick="Submit Click" />
<asp:Label ID="subLabel" runat="server" Text="您的提交："></asp:Label>
<asp:Panel ID="Panel1" runat="server">正确答案：</asp:Panel>
```

（2）接下来定义【提交】按钮的鼠标单击事件，遍历学生的答案并显示，同时添加 Literal 控件显示正确答案，代码如下。

```
protected void Submit Click(object sender, EventArgs e)
{
    string cla = classList.SelectedValue;
    string dir = direction.SelectedValue;
    string provinces = "";
    foreach (ListItem item in province.Items)          //遍历集合项
    {
        if (item.Selected)                             //某项是否选中
        { provinces += item.Value + "、"; }
    }
    if (cla == "" || dir == "" || provinces == "")
    {
        subLabel.Text = "请填写完整";
        subLabel.ForeColor = System.Drawing.Color.Red;
    }
    else
    {
        subLabel.Text = "您的提交：" + cla + "；  您的答案是：" + dir + "；" +
        provinces;
        subLabel.ForeColor = System.Drawing.Color.Black;
        Literal liter = new Literal();
        liter.ID = "literText";
        liter.Text = "南；   黑龙江、吉林、辽宁";
        Panel1.Controls.Add(liter);
    }
}
```

（3）运行该页面，选择【三年级】并选中一些答案，单击【提交】按钮，页面的运行效果如图 3-7 所示。

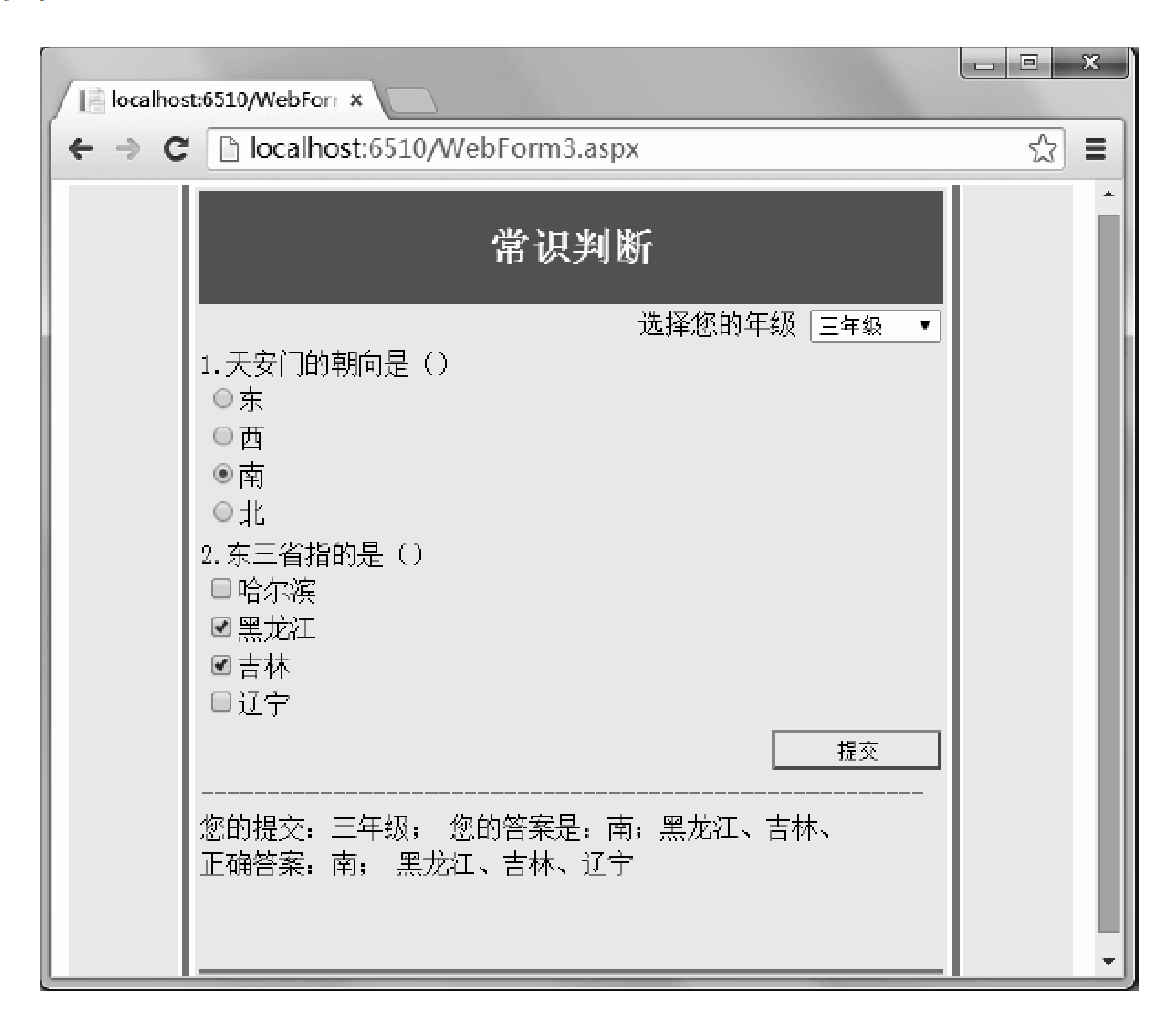


图 3-7 常识调查页面

思考与练习

一、填空题

1．Button 控件的__________属性表示与控件相关联的命令名。

2. CheckBox 控件的__________属性表示某一项是否被选取。

3. 如果 Image 控件设置的图像路径不存在，可以通过__________属性设置无法显示图像时的文本。

4. 将 ListBox 控件的 SelectionMode 属性的值设置为__________时允许用户选择多个项目。

二、选择题

1．与 HyperLink 外观和功能类似的控件是__________。

A．LinkImage

B．LinkLabel

C．LinkButton

D．LinkList

2．设置按钮单击事件的属性是_________。

A．OnClick

B．On Checked

C．Click

D．ClickOn

3．判断 CheckBoxList 中的选项是否被选中使用__________属性。

A．Checked

B．Selected

C．IsCheck

D．IsSelect

4．当用户修改 DropDownList 控件当前项时引发该控件的__________事件。

A．SelectedChanged

B．IndexChanged

C．ChangedSelected

D．SelectedIndexChanged

三、简答题

1．简述控件的分类。

2．概括服务器控件的特点。

3．概括 RadioButtonList 和 RadioButton 控件的区别。

4．概括 RadioButton 和 CheckBox 控件的区别。

第 4 章　页面请求与响应对象

尽管 ASP.NET 采用事件响应模式，但是它毕竟是 Web 应用程序，因此，ASP.NET 无论如何都无法脱离 B/S 程序结构的运行原理。ASP.NET 的出现改变了传统 Web 开发的许多习惯，但是开发者在实现某些操作的过程中，还是会用到 ASP.NET 提供的一些内置对象。例如，从一个页面 Login.aspx 跳转到另一个页面 Success.aspx 时用到 Response 对象；接收从上个页面传递的参数的值用到 Request 对象。

本章重点介绍 ASP.NET 提供的页面请求与响应对象，这些对象分别是 Page、Response、Request 和 Server。

本章学习要点：

- ❑ 掌握 Page 对象的常用属性
- ❑ 熟悉 Page 对象的方法和事件
- ❑ 熟悉 Response 对象的常用属性
- ❑ 掌握 Response 对象的常用方法
- ❑ 掌握 Request 对象的常用属性
- ❑ 了解 Request 对象的常用方法
- ❑ 了解 Server 对象的常用属性
- ❑ 掌握 Server 对象的常用方法

4.1 Page 对象

严格来说，Page 对象是一个页面级别的对象，该对象是由 System.Web.UI 命名空间中的 Page 类来实现的，作为公有属性被声明在 System.Web.UI.Control 类中，它被 System.Web.UI.TemplateControl 类继承。

4.1.1 Page 对象的属性

每一个 ASP.NET 页面都会对应一个页面类，Page 对象就是页面类的实例。因为，Page 对象是由 System.Web.UI 命令空间中的 Page 类来实现，因此，Page 类与扩展名为“.aspx”的文件相关联，这些文件在运行时被编译为 Page 对象，并缓存在服务器内存中。

Page 对象提供多个属性，通常这些属性可以获取不同的内容，常用属性如表 4-1 所示。

表 4-1　Page 对象的常用属性

属性名称	说明
IsPostBack	获取一个值，该值指示该页是否正为响应客户端回发而加载
IsValid	获取一个值，该值表示页面是否通过验证

续表

属性名称	说明
Application	为当前 Web 请求获取 Application 对象
Request	获取请求的页的 HttpRequest 对象
Response	获取与 Page 关联的 HttpResponse 对象。该对象使开发者能够将 HTTP 响应数据发送到客户端，并包含有关该响应的信息
Session	获取 ASP.NET 提供的当前 Session 对象
Server	获取 Server 对象，它是 HttpServerUtility 类的实例

在表 4-1 中列出了 Page 对象的多个属性，下面简单介绍 IsPostBack 属性和 IsValid 属性。

1. IsPostBack 属性

IsPostBack 属性用来获取一个布尔值，如果返回值为 true，则表示当前页是为响应客户端回发；如果返回值为 false，则表示当前页是首次加载。

【范例 1】

在 Web 窗体页中分别添加一个用户名输入框、一个密码框和一个提交按钮，代码如下。

```
Name: <asp:TextBox ID="txtName" runat="server"></asp:TextBox><br /><br />
Pass: <asp:TextBox ID="txtPass" TextMode="Password" runat="server">
</asp:TextBox><br /><br />
<asp:Button ID="Button1" runat="server" Text="Button" />
```

在 Web 窗体页后台 Load 事件中添加代码，通过 Page.IsPostBack 判断页面是否为首次加载，如果是则将用户名输入框的值显示为 admin，否则显示“回发”，代码如下。

```
protected void Page Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack) {                      //页面首次加载
        txtName.Text = "admin";
    } else {                                     //页面回发
        txtName.Text = "回发";
    }
}
```

运行窗体页面查看效果，如图 4-1 所示为首次加载时的效果。直接单击 Button 按钮进行测试，如图 4-2 所示。

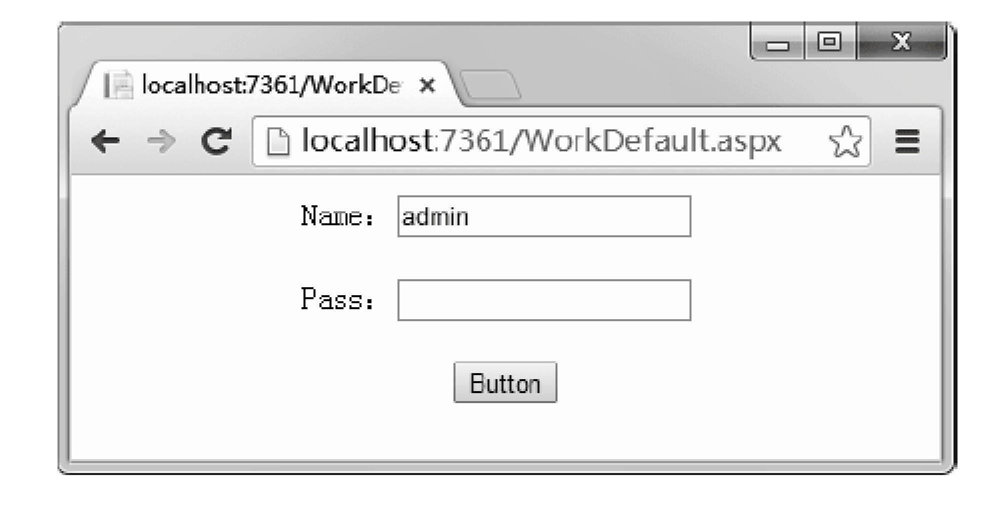


图 4-1 页面首次加载

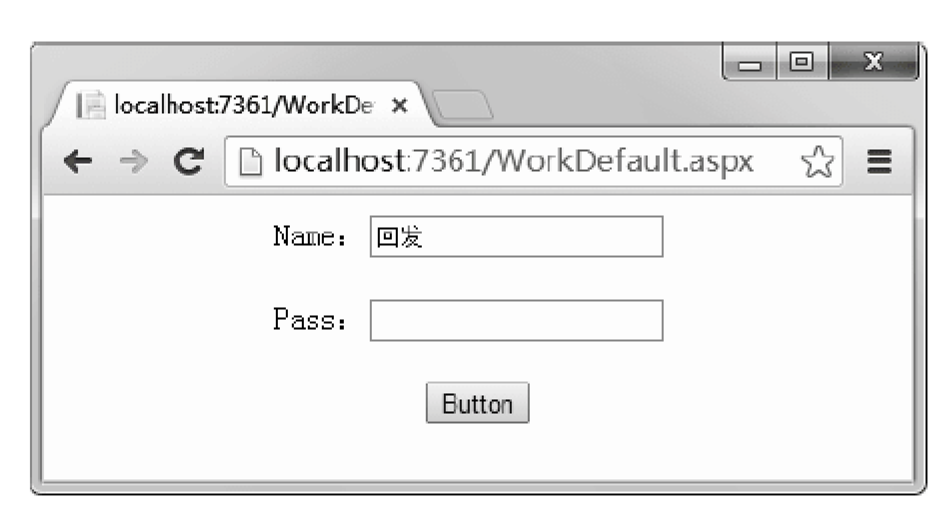


图 4-2 页面回发加载

2. IsValid 属性

IsValid 属性用来获取一个布尔值，该值指示页验证是否成功。如果页验证成功，则为 true；否则为 false。一般在包含验证服务器控件的页面中使用，只有在所有验证服务器控件都验证成功时，IsValid 属性的值才为 true。

【范例 2】

在范例 1 的基础上添加代码，实现步骤如下。

（1）为用户输入框和密码框添加 RequiredFieldValidator 验证控件，必须保证用户输入登录名和密码，代码如下。

```
Name: <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvName" runat="server" ControlToValidate=
"txtName" ErrorMessage="必须填写"></asp:RequiredFieldValidator><br /><br />
Pass: <asp:TextBox ID="txtPass" TextMode="Password" runat="server">
</asp:TextBox>
<asp:RequiredFieldValidator ID="rfvPass" runat="server" ControlToValidate=
"txtPass" ErrorMessage="必须填写"></asp:RequiredFieldValidator><br />
<br />
```

（2）为窗体中的 Button 控件添加 Click 事件。页面代码如下。

```
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_
Click" />
```

（3）在 Button 控件之后添加一个用于显示信息的 Label 控件。窗体代码如下。

```
<asp:Label ID="lblResult" runat="server" ></asp:Label>
```

（4）在窗体页面后台添加 Button 控件的 Click 事件代码，通过 Page.IsValid 属性判断是否通过验证。内容如下。

```
protected void Button1 Click(object sender, EventArgs e)
{
    if (Page.IsValid) {           //如果输入的信息通过验证
        lblResult.Text = "通过验证";
    }
}
```

（5）运行窗体页面查看效果，直接单击图中的 Button 按钮测试，如图 4-3 所示。在页面中输入用户名和密码后再次单击 Button 按钮，如图 4-4 所示。

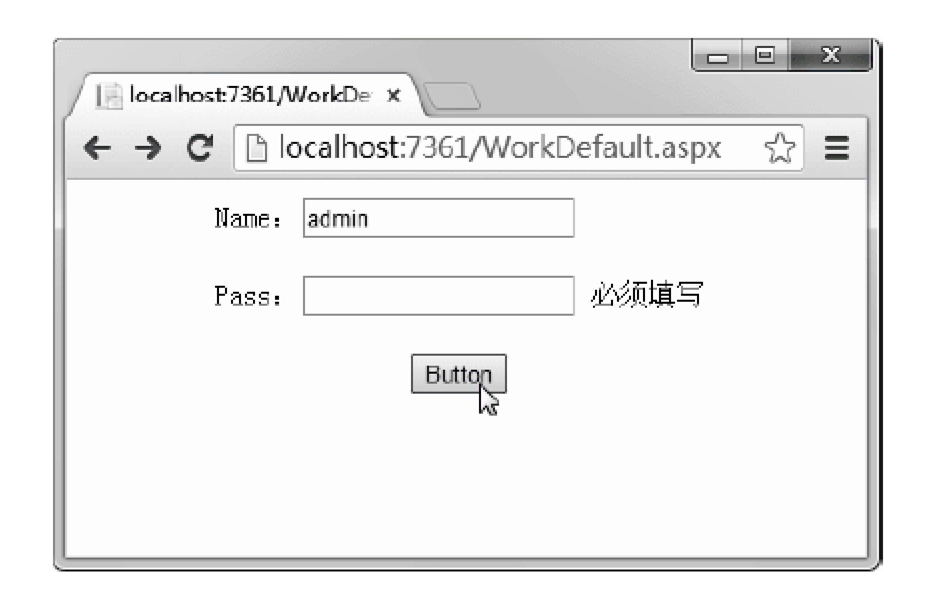


图 4-3 没有通过验证

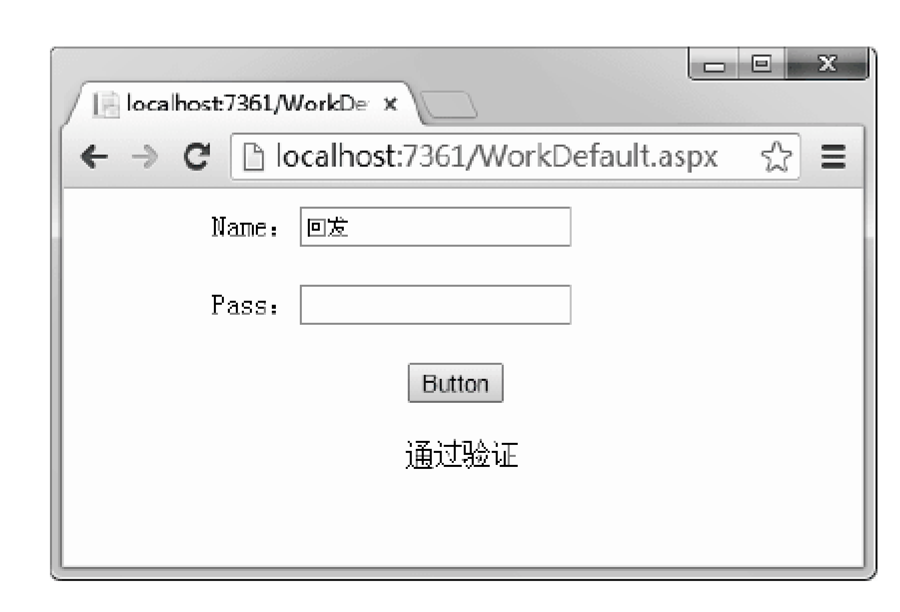


图 4-4 已经通过验证

4.1.2 Page 对象的方法

Page 对象最常用的就是属性，但是它也提供了一系列的方法和事件，最常用的方法有以下两个。

（1）DataBind()方法：将数据源绑定到被调用的服务器控件及其所有子控件。

（2）RegisterClientScriptBlock()方法：向页面发出客户端脚本块。

使用 Page.RegisterClientScriptBlock()方法可以在页面中输出脚本，但是这种方式已经过时。实际上，Page 对象支持 ClientScript 属性，该属性返回 ClientScriptManager 对象。调用 ClientScriptManager 对象提供的方法也可以输出脚本，如表 4-2 所示。

表 4-2 ClientScriptManager 对象的常用方法

方法名称	说明
RegisterArrayDeclaration()	添加一个 JavaScript 数组到页面中
RegisterClientScriptBlock()	在页面的起始<form>标记后添加 JavaScript 脚本
RegisterClientScriptInclude()	在页面的起始服务端<form>标签后添加外部 JavaScript 文件引用
RegisterClientScriptResource()	添加已编译到程序集中的 JavaScript 资源文件的链接到页面中
RegisterExpandAttribute()	生成为页面中的元素附加扩展属性的脚本
RegisterForEventValidation()	生成事件验证代码，事件验证指事件
RegisterHiddenField()	在页面的起始服务端<form>标签后添加一个隐藏表单域
RegisterOnSubmitStatement()	用于添加在页面回传服务端前执行的 JavaScript 脚本
RegisterStartupScript()	向页面对象注册启动脚本

RegisterClientScriptBlock()方法输出脚本信息时，可以将 JavaScript 函数放在页面的顶部。也就是说，该脚本用于在浏览器中启动页面。两个形式如下。

```
void ClientScriptManager.RegisterClientBlock(Type type, string key,
string script)
void ClientScriptManager.RegisterClientBlock(Type type, string key,
string script, bool addScriptTags)
```

其中，type 表示要注册的客户端脚本的类型；key 表示要注册的客户端脚本的键；script 表示要注册的客户端脚本文本；addScriptTags 表示是否添加脚本标记的布尔值。

【范例 3】

更改范例 2 的代码，重新为 Button 控件的 Click 事件添加代码。在该事件代码中判断用户输入的用户名和密码是否都为 admin，如果是则弹出“恭喜您，输入的用户名和密码正确！”对话框提示，否则弹出“很抱歉，用户名或者密码错误！”对话框提示。代码如下。

```
if (txtName.Text == "admin" && txtPass.Text == "admin") {
    Page.ClientScript.RegisterClientScriptBlock(GetType(), "", "<script>
    alert('恭喜您，输入的用户名和密码正确！')</script>");
} else {
    Page.ClientScript.RegisterClientScriptBlock(GetType(), "", "<script>
    alert('很抱歉，用户名或者密码错误！')</script>");
}
```

运行窗体页面输入用户名和密码后单击按钮测试，如图 4-5 所示。

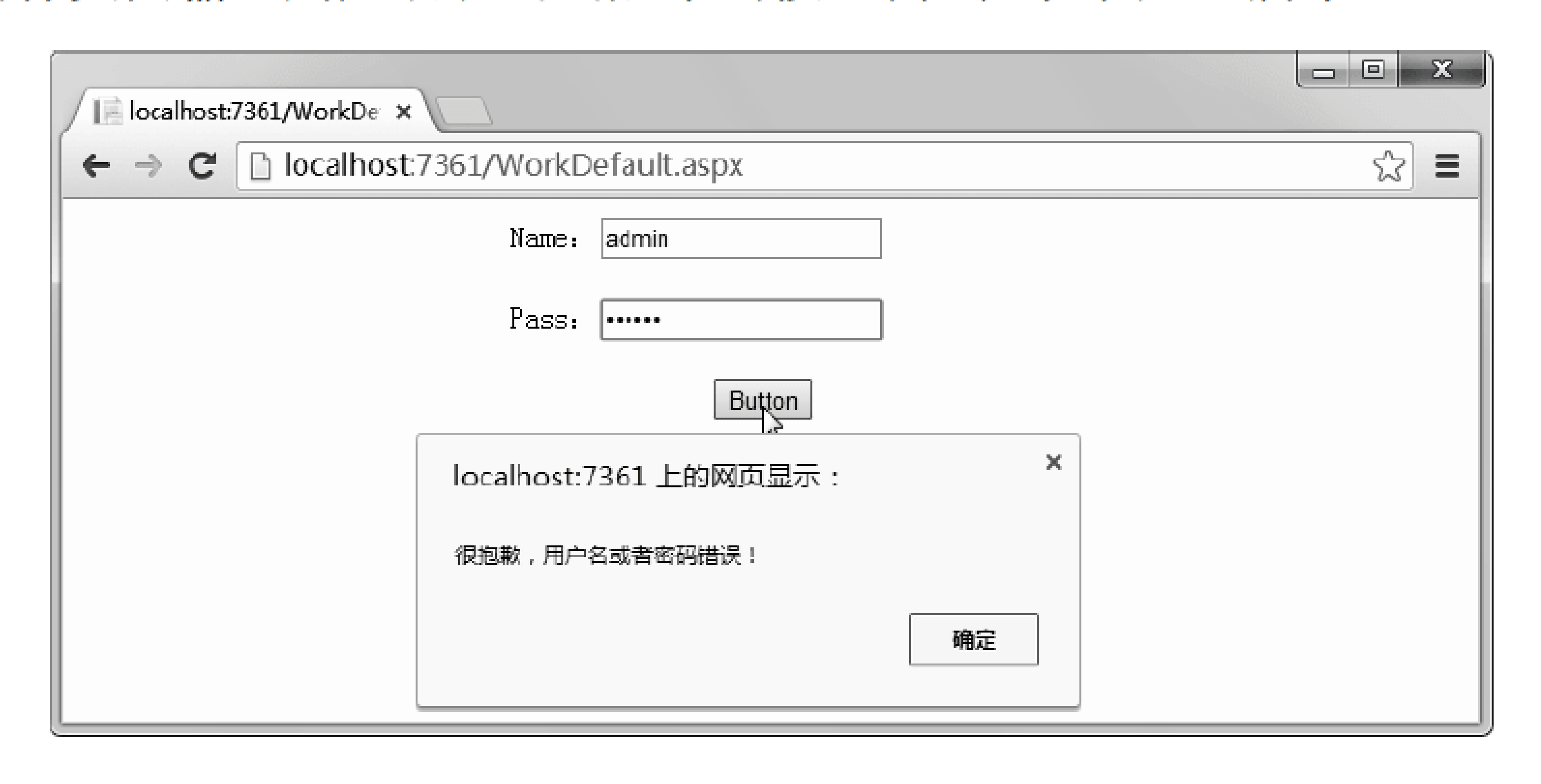


图 4-5　RegisterClientScriptBlock()方法的使用

提 示

除了 RegisterClientBlock()方法外，RegisterStartupScript()方法也可以输出脚本。这两个方法的区别不大，最大的区别就是 RegisterStartupScript()方法把脚本放在 ASP.NET 页面的底部，而不是顶部。实际上，RegisterStartupScript()方法与 RegisterClientBlock()方法具有相同的构造函数，开发者在查找、调用页面控件时可用该方法。

4.1.3　Page 对象的事件

Page 对象包含一系列的事件，常用的三个事件说明如下。

（1）Init 事件：当服务器控件初始化时发生。

（2）Load 事件：当服务器控件加载到 Page 对象中时发生。

（3）Unload 事件：当服务器控件从内存中卸载时发生。

4.2　Response 对象

页面跳转与数据传递是 ASP.NET 中页面最常执行的操作，页面跳转是指从一个页面跳转到另一个页面。要实现跳转，可以用 ASP.NET 的 Response 对象。

Response 对象提供对当前页面的输出流的访问，这里的输出流是指用户作为对其请求的响应而收到的信息集合。

4.2.1　Response 对象的属性

Response 对象主要可用于：将文本写入到输出页面；读取/写入 Cookie；将用户从请求页面重新定向到另一页面；结束基于某些条件的应用程序连接；检查客户端是否依然与服务器相连等功能。

实际上，Response 对象是 HttpResponse 类的一个对象，与一个 HTTP 响应相对应，通过该对象的属性和方法可以控制如何将服务器端的数据发送到客户端浏览器。如表 4-3 所示列出了 Response 对象的常用属性，并对这些属性进行说明。

表 4-3 Response 对象的常用属性

属性名称	说明
Cookies	获取响应 Cookie 集合
Buffer	获取或设置一个值，该值指示是否缓冲输出，并在完成处理整个响应之后将其发送
BufferOutput	获取或设置一个值，该值指示是否缓冲输出，并在完成处理完整个页之后将其发送
Expires	获取或设置在浏览器上缓存的页过期之前的分钟数。如果用户在页过期之前返回同一页，则显示缓存的版本
Output	启用到输出 HTTP 响应流的文本输出
OutputStream	启用到输出 HTTP 内容主体的二进制输出
Charset	获取或设置输出流的 HTTP 字符集
IsClientConnected	获取一个值，通过该值指示客户端是否仍连接在服务器上
ContentEncoding	获取或设置输出流的 HTTP MIME 类型
SuppressContent	是否将 HTTP 内容发送到客户端
ContentType	获取或设置输出流的 HTTP MIME 类型

通过 Response 的 ContentType 属性设置页面的内容类型为 JPEG 文件，并且将 BufferOutput 属性的值设置为 true，表示缓冲响应，以便发送页面，代码如下。

```
Response.ContentType = "image/jpeg";
Response.BufferOutput = true;
```

4.2.2 Response 对象的方法

Response 对象可以输出信息到客户端，包括直接发送消息给浏览器、重定向浏览器到另一个 URL 或设置 Cookie 的值。ASP.NET 中引用对象方法的语法是“对象名.方法名”。Response 对象提供了多个方法，常用方法如表 4-4 所示。

表 4-4 Response 对象的常用方法

方法名称	说明
AddHeader()	将一个 HTTP 标头添加到输出流
BinaryWrite()	将一个二进制字符串写入 HTTP 输出流
Clear()	清除缓冲区流中的所有内容输出
End()	将当前所有缓冲的输出发送到客户端，停止该页的执行，并引发 EndRequest 事件
Redirect()	将客户端重定向到新的 URL
Write()	将信息写入 HTTP 响应输出流
WriteFile()	将指定的文件直接写入 HTTP 响应输出流

1. Write()方法

Write()方法将信息写入 HTTP 响应输出流。Write()方法有以下 4 种构造函数。

```
void HttpResponse.Write(char ch)
void HttpResponse.Write(object obj)
void HttpResponse.Write(string s)
void HttpResponse.Write(char[] buffer, int index, int count)
```

其中，第一种构造函数表示将一个字符写入 HTTP 响应输出流，ch 表示要写入 HTTP 输出流的字符。第二种构造函数表示将 System.Object 写入 HTTP 响应流，obj 表示要写入 HTTP 输出流的 System.Object。第三种构造函数表示将一个字符串写入 HTTP 响应输出流，s 表示要写入 HTTP 输出流的字符串。最后一种构造函数表示将一个字符数组写入 HTTP 响应输出流，buffer 表示要写入的字符数组。

【范例 4】

通常情况下，开发者会使用 Write()方法向页面输出一段脚本或者一段文本内容。如在 Web 窗体中设计用户登录页面，包含一个用户输入框、一个密码框、一个按钮和两个必填验证控件，并为按钮添加 Click 事件。Click 事件代码如下。

```
protected void Button1 Click(object sender, EventArgs e) {
    string content = "您输入的用户名是：" + txtName.Text + "，您输入的密码是：
    " + txtPass.Text;
    Response.Write(content);
}
```

运行窗体页面查看初始效果，如图 4-6 所示。输入内容后单击 Button 按钮测试，如图 4-7 所示。

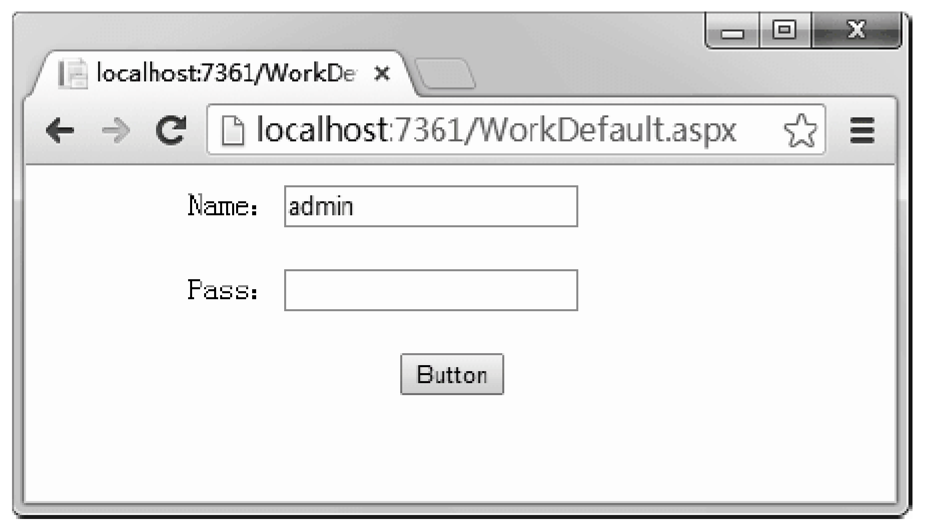


图 4-6 初始效果

图 4-7 输出文本

通过 Write()方法输出脚本的实现也很简单，直接将脚本作为参数传递到 Write()方法中即可，如范例 5 所示。

【范例 5】

更改范例 4 的代码，通过 Write()方法弹出范例 4 输出的文本，代码如下。

```
protected void Button1 Click(object sender, EventArgs e) {
    string content = "您输入的用户名是：" + txtName.Text + "，您输入的密码是：
    " + txtPass.Text;
    Response.Write("<script>alert('"+content+"')</script>");
}
```

运行窗体页面，输入内容后单击 Button 按钮测试，如图 4-8 所示。

图 4-8　Write()方法输出脚本

2．WriteFile()方法

Response 对象提供了一个直接输出文本文件的 WriteFile()方法。如果要输出的文件和执行的网页在同一个目录，只要直接传入文件名称即可；如果不在同一个目录，则要指定详细的目录名称。

【范例 6】

页面加载时直接调用 Response 对象的 WriteFile()方法输出 Test.txt 文本文件中的内容，代码如下。


```
protected void Page_Load(object sender, EventArgs e) {
    Response.WriteFile("Test.txt");
}
```

3．Redirect()方法

Response 对象的 Redirect()方法将客户端浏览器重定向到另外的 URL 上，即跳转到另一个网页。Redirect()方法的两种常用构造函数如下。

```
void HttpResponse.Redirect(string url)
void HttpResponse.Redirect(string url, bool endResponse)
```

上述两种构造函数都表示跳转到指定的 URL。其中，第一种方式需要传入一个参数，用 URL 指定路径，使请求自动跳转到指定的 URL 地址。第二种方式需要传入两个参数，用 URL 指定路径，并指定一个布尔值指示当前响应是否结束。

【范例 7】

在 Button 控件的 Click 事件中添加代码，获取用户在页面中输入的用户名和密码，如果用户名和密码都是 admin，则跳转到 LoginSuccess.aspx 页面，否则跳转到 LoginFail.aspx 页面，代码如下。

```
protected void Button1_Click(object sender, EventArgs e) {
    if (txtName.Text == "admin" && txtPass.Text == "admin") {
        Response.Redirect("LoginSuccess.aspx");
    } else {
```

```
            Response.Redirect("LoginFail.aspx");
        }
    }
```

在用 Redirect()方法实现页面跳转时，可以通过问号（?）向页面中传递参数。如向 LoginSuccess.aspx 页面中传入 name 和 pass 两个参数，代码如下。

```
    Response.Redirect("LoginSuccess.aspx?name="+txtName.Text+"&&pass="+txt
    Pass.Text);
```

4.3 Request 对象

在 B/S 结构的应用程序中，客户端要向服务器端发出请求，这些请求的信息包括客户端信息、请求的 URL、请求的参数和 Cookie 等内容，服务器在接收到用户请求时自动将请求封装到 Request 对象中供开发者使用。

4.3.1 Request 对象的属性

Request 对象用来使服务器取得客户端浏览器的数据信息。它实际上是 System.Web 命名空间中的 HttpRequest 类的对象。当客户发出请求执行 ASP.NET 程序时，客户端的请求信息会包装在 Request 对象中，这些请求信息包括请求报头（Header）、客户端的机器信息、客户端浏览器信息、请求方法（如 POST 和 GET）和提交的窗体信息等。

Request 对象的属性和方法相当多，如表 4-5 所示对常用的属性进行了说明。

表 4-5 Request 对象的常用属性

属性名称	说明
Buffer	获取或设置一个值，该值指示是否缓冲输出并在处理完整个响应之后发送它
Browser	获取或者设置有关正在请求的客户端的浏览器功能的信息
Cookies	获取客户端发送的 Cookie 的集合
ContentLength	指定客户端发送的内容长度
Form	获取窗体变量集合
Files	获取客户端上传的文件集合。如果该属性的值为 HttpFileCollection 对象，表示客户端上传的文件集合
FilePath	获取当前请求的虚拟路径
QueryString	获取 HTTP 查询字符串变量集合
UserHostAddress	获取远程客户端的 IP 主机地址
UserHostName	获取远程客户端的 DNS 名称
UserLanguages	获取客户端语言首选项的排序字符串数组
UserAgent	获取客户端浏览器的原始用户代理信息
Params	获取 QueryString、Form、ServerVariables 和 Cookies 项的组合集合
ServerVariables	获取 Web 服务器变量的集合
Url	获取有关当前请求的 URL 的信息

在表 4-5 中列出了多个属性，下面只介绍常用的一些属性，如 Browser 属性、Form 属性和 QueryString 属性等。

1. Browser 属性

Browser 属性获取有关正在请求的客户端的浏览器功能的信息，该属性的值是一个 HttpBrowserCapabilities 对象。如表 4-6 所示对 HttpBrowserCapabilities 对象的常用属性进行说明。

表 4-6 HttpBrowserCapabilities 对象的常用属性

属性名称	说明
Type	获取客户端浏览器的名称和主要版本号
Browser	获取客户端浏览器的名称
Version	获取客户端浏览器的版本
Platform	获取客户端使用的操作平台的名称
Frames	获取客户端浏览器是否支持框架
Cookies	获取客户端浏览器是否支持 Cookies
JavaScript	获取客户端浏览器是否支持 JavaScript

【范例 8】

在 Web 窗体中添加一个 Label 控件，该控件向页面输出信息，代码如下。

```
<asp:Label ID="lblMessage" runat="server"></asp:Label>
```

在窗体页面后台的 Load 事件中指定 Label 控件的 Text 属性值，代码如下。

```
protected void Page_Load(object sender, EventArgs e) {
    lblMessage.Text = "浏览器的类型是：" + Request.Browser.Browser + "<br>"
        + "浏览器的版本是：" + Request.Browser.Version + "<br>"
        + "浏览器的所在平台是：" + Request.Browser.Platform + "<br>"
        + "浏览器是否支持框架：" + Request.Browser.Frames + "<br>"
        + "浏览器是否支持 Cookies：" + Request.Browser.Cookies + "<br>"
        + "浏览器是否支持 JavaScript：" + Request.Browser.JavaScript + "<br>";
}
```

运行窗体页面查看输出结果，如图 4-9 所示。



图 4-9 Browser 属性的使用

2. Form 属性

Form 属性可以检索发送到 HTTP 语法中控件的值，来实现信息的提交和处理。使用 Form 属性时需要注意，页面表单提交的方式要设置为 POST，与 GET 相比较，POST 可

以将大量数据发送到服务器端。

【范例 9】

通过 Request 对象的 Form 属性获取用户提交的表单的数据，并且判断提交的用户和密码是否为 admin。实现步骤如下。

（1）创建 FormDefault.aspx 窗体页面，指定表单元素的 method 属性值为 post，并在表单中添加一个 3 行 2 列的表格，第一行提供用户名输入框，第二行提供密码输入框，最后一行显示操作按钮，代码如下。

```
<form id="form1" runat="server" method="post">
    <table width="100%" height="100px">
        <tr><td align="right">登录名：</td><td><asp:TextBox ID="txtName"
        runat="server"></asp:TextBox></td></tr>
        <tr><td align="right">登录密码：</td><td><asp:TextBox ID="txtPass"
        runat="server" TextMode="Password"></asp:TextBox></td></tr>
        <tr><td></td><td><asp:Button ID="btnSubmit" runat="server" Text="
        登 录" PostBackUrl="~/RequestTest/MessageDefault.aspx" /></td>
        </tr>
    </table>
</form>
```

（2）创建 MessageDefault.aspx 窗体页面，在该页面中添加用于显示结果的 Label 控件。

（3）在 MessageDefault.aspx.cs 后台页面的 Load 事件中添加代码，通过 Request.Form 属性获取 FormDefault.aspx 窗体页面控件的值，并判断获取的值是否都等于 admin，代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    string name = Request.Form["txtName"].ToString();
    string pass = Request.Form["txtPass"].ToString();
    if (name == "admin" && pass == "admin") {
        lblLoginMessage.Text = "用户名：" + name + "，密码：" + pass + "登
        录成功。";
    } else {
        lblLoginMessage.Text = "用户名：" + name + "，密码：" + pass + "登
        录失败。";
    }
}
```

（4）运行 FromDefault.aspx 页面并输入内容，如图 4-10 所示。输入内容完毕后单击【登录】按钮，如图 4-11 所示。



图 4-10 登录页面



图 4-11 登录效果

注 意

通过 Response 对象的 Redirect()方法跳转页面时，源页面保存的所有数据信息将丢失。因此，目标页面无法访问源页面提交的数据，但是可以通过其他方式（如 Session 对象）实现。本范例中通过 Button 控件的 PostBackUrl 属性实现页面跳转。

3．QueryString 属性

QueryString 属性与 Form 属性不同，它用于获取 HTTP 查询字符串变量集合。简单来说，QueryString 属性用来获取以 GET 方式提交的表单数据。使用该属性时，可以通过键名或者索引两种方式获取，语法如下。

```
Request.QueryString[0];
Request.QueryString["key"]
```

【范例 10】

利用范例 9 的页面，通过 QueryString 属性获取用户传入的信息。主要步骤如下。

（1）在 QueryStringDefault.aspx 页面中设计登录信息，为页面中的 Button 控件指定 Click 事件，而不是 PostBackUrl 属性。

（2）在 QueryStringDefault.aspx 页面的后台 Load 事件中添加以下代码。

```
protected void btnSubmit Click(object sender, EventArgs e) {
    Response.Redirect("MessageDefault.aspx?name=" + txtName.Text + "&
    pass=" + txtPass.Text);
}
```

（3）更改 MessageDefault.aspx 页面的后台 Load 事件中的代码，通过 QueryString 属性的索引方式获取从页面传递的参数信息，代码如下。

```
string name = Request.QueryString[0].ToString();
string pass = Request.QueryString[1].ToString();
```

（4）运行 QueryStringDefault.aspx 窗体页面进行测试，效果不再显示。

4．Params 属性

无论以哪种方式（如 POST 或 GET）提交表单数据，如果用错（如 GET 方式提交表单，Form 属性取值）就获取不到信息，这时可以通过 Params 属性获取数据。Request.Params 是所有 POST 和 GET 传过来的值的集合，它依次包括 Request.QueryString、Request.Form、Request.Cookies 和 Request.ServerVariable。基本语法如下。

```
Request.Params["key"]
```

5．ServerVariables 属性

Request 对象用 ServerVariables 属性保存服务器端系统信息，这些信息变量包含在 HTTP 头部中随 HTTP 请求一起传送。ServerVariables 的使用语法如下。

```
Request.ServerVariables["服务器变量名称"]
```

ServerVariables 属性中保存多个信息变量，表 4-7 列出了常用的变量名称。

表 4-7 ServerVariables 中保存的常用变量名称

服务器变量名称	说明
ALL_HTTP	客户端浏览器发送的 HTTP 头部
ALL_RAW	获取传送给浏览器的原始数据
ALL_MD_PATH	Web 应用程序的相对路径
ALL_PHYSICAL_PATH	Web 应用程序的物理路径
LOCAL_ADDR	获取要求的服务器地址
PATH_INFO	ASP.NET 程序的相对路径信息
REMOTE_ADDR	浏览器所在主机的 IP 地址
REMOTE_HOST	浏览器所在主机的计算机名
SCRIPT_NAME	正在运行的脚本的名称
SERVER_NAME	运行脚本的服务器的主机名、DNS 或 IP 地址
SREVER_PORT	获取服务器端口号
SERVER_PROTOCOL	获取通信协议的名称以及编号
SERVER_SOFTWARE	获取服务器端软件的名称及版本

如果要查看 ServerVariables 属性中保存的所有信息变量名称，可以通过 foreach 语句遍历，代码如下。

```
foreach (string str in Request.ServerVariables.AllKeys) {
    Response.Write(str+"<Br/>");
}
```

提 示

除上述介绍的属性外，Request 对象还会用到其他属性获取信息，如 Url 获取当前请求的 URL 地址，UserHostAddress 获取客户端的 IP 主机地址。这些属性的使用都很简单，这里不再一一列举。

4.3.2 Request 对象的方法

Request 对象提供一系列的方法，但是并非所有的方法都会被用到。其中，BinaryRead()方法用来对当前输入流进行指定字节数的二进制读取。MapPath()方法将当前请求的 URL 中的虚拟路径映射到服务器上的物理路径。

MapPath()方法接收一个字符串类型参数，并通过传入的字符串获取相应的物理路径。例如，页面加载时获取当前目录下 FormDefault.aspx 页面的完整路径，代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    Response.Write(Request.MapPath("FormDefault.aspx"));
}
```

4.4 Server 对象

Server 对象也是 Page 对象的成员之一，主要提供一些处理网页请求时所需的功能，例如建立 COM 对象、将字符串进行编码和解码等工作。

4.4.1 Server 对象的属性

Server 对象提供对服务器上的方法和属性的访问，它是 HttpServerUtility 类的实例。Server 对象提供两个属性，说明如下。

（1）MachineName 属性：获取服务器的计算机名称。

（2）ScriptTimeout 属性：获取和设置请求超时（以秒计）的时间。

例如，在窗体页面上添加 Label 控件，该控件显示计算机名称和请求超时的时间。Load 事件中的代码如下。

```
protected void Page_Load(object sender, EventArgs e) {
    lblResult.Text = "计算机名称：" + Server.MachineName + "<br/>获取超时的
    时间：" + Server.ScriptTimeout;
}
```


4.4.2 Server 对象的方法

与 Server 对象的属性相比，它的方法要多一些，常用方法如表 4-8 所示。

表 4-8 Server 对象的常用方法

方法名称	说明
CreateObject()	创建 COM 对象的一个服务器实例
CreateObjectFromClsid()	创建 COM 对象的服务器实例，该对象由对象的类标识符标识
Transfer()	用于终止当前页的执行，并为当前请求开始执行新页
MapPath	返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径
HtmlDecode()	对已被编码以消除无效 HTML 字符的字符串进行解码
HtmlEncode()	对要在浏览器中显示的字符串进行编码
UrlEncode()	编码字符串，以便通过 URL 进行可靠的 HTTP 传输
UrlDecode()	对字符串进行解码
Execute()	执行当前服务器上的另一个窗体页面，执行完该页面后再返回本页继续执行

虽然表 4-8 中列出了 Server 对象的多个方法，但是下面只对该对象的常用方法进行介绍。

1. MapPath()方法

Server 对象的 MapPath()方法返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径。使用该方法时需要传入一个 string 类型的参数，该参数是虚拟路径（即相对路

径），返回传入的相对路径的绝对路径。如果传入参数的值是 null，则返回应用程序所在的目录的物理路径。

【范例 11】

页面加载时指定 ID 属性值为 lblResult 的 Label 控件的 Text 属性值，代码如下。

```
protected void Page_Load(object sender, EventArgs e){
    lblResult.Text = "相对地址：" + Server.MapPath("images")+"<br/>"
        + "指定~符号：" + Server.MapPath("~/images")+"<br/>"
        + "传入null：" + Server.MapPath(null);
}
```

运行窗体页面查看效果，如图 4-12 所示。

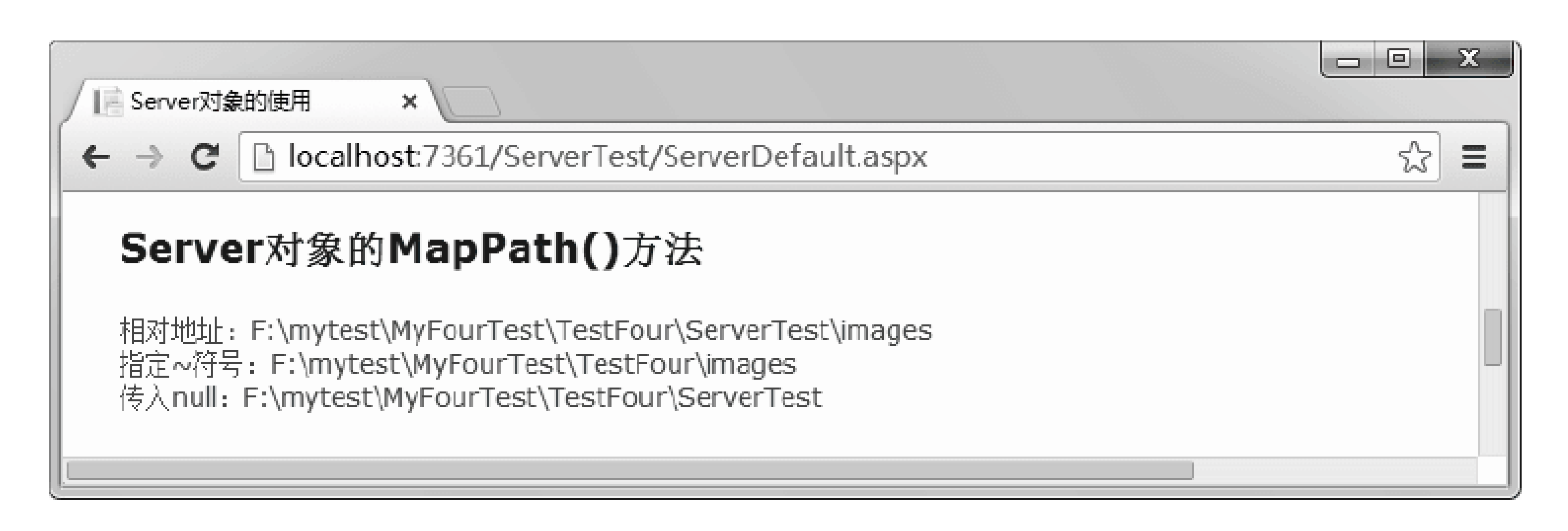


图 4-12 MapPath()方法的使用

2．Transfer()和 Execute()方法

实现从一个页面 A.aspx 跳转到另一个页面 B.aspx，常用的有以下 5 种方法。

（1）通过 Response 对象的 Redirect()方法实现。

（2）通过 Response 对象的 Write()方法实现，在该方法中通过脚本跳转页面。

（3）通过 Button 控件、ImageButton 控件和 LinkButton 控件的 PostBackUrl 属性实现。

（4）通过 Server 对象的 Transfer()方法实现。

（5）通过 Server 对象的 Execute()方法实现。

通过前面的三种方式实现页面跳转时，地址栏中的页面都会发生改变。例如，范例 9 通过 PostBackUrl 属性实现跳转，当单击页面中的【登录】按钮时会提交到 MessageDefault.aspx 页面，并且地址栏中的页面发生改变。但是，使用 Server 对象的 Transfer()方法实现跳转时，地址栏中的页面不会发生改变。

【范例 12】

下面演示 Server 对象的 Transfer()方法的使用。

（1）创建 Transfer.aspx 窗体页面，并在该页面中添加一个 Button 控件。页面代码如下。

```
<asp:Button ID="btn1" runat="server" Text="Transfer()" OnClick=
"btn1_Click" />
```

（2）在后台页面的 Load 事件中添加如下代码。

```
protected void btn1_Click(object sender, EventArgs e) {
```

```
    Server.Transfer("Default.aspx");
}
```

（3）创建 Default.aspx 页面，该页面的 HTML 文档中只包含一段文本，内容是“这是一个简单的 ASP.NET 网页。”。

（4）运行窗体页面查看效果，如图 4-13 所示。单击图中的按钮进行测试，如图 4-14 所示。比较图 4-13 和图 4-14 可以发现，虽然实现页面跳转，但是地址栏并没有发生改变。



图 4-13　页面初始效果



图 4-14　实现页面跳转

通过 Server 对象的 Execute()方法也可以实现从 A.aspx 页面到 B.aspx 页面的跳转，但是不同的是，执行完 B.aspx 页面之后会重新返回到 A.aspx 页面继续执行。

【范例 13】

在范例 12 的基础上添加新的内容，步骤如下。

（1）在创建 Transfer.aspx 窗体页面中添加文本值为“Execute()”的 Button 控件，并指定控件的 Click 事件。

（2）为上述添加的 Button 控件添加 Click 事件代码，调用 Execute()方法实现跳转。

（3）运行窗体页面查看效果，如图 4-15 所示。单击 Execute()按钮进行测试，如图 4-16 所示。



图 4-15　页面初始效果



图 4-16　实现跳转效果

3．HtmlEncode()和 HtmlDecode()方法

HtmlEncode()方法对字符串进行编码；HtmlDecode()方法对字符串进行解码。HtmlEncode()和 HtmlDecode()方法都有两种构造函数，以 HtmlEncode()方法为例，形式如下。

```
string HtmlEncode(string s)
void HtmlEncode(string s, TextWriter output)
```

其中，第一种形式的构造函数表示对字符串进行 HTML 编码并返回已编码的字符串，参数 s 是要编码的字符串。第二种形式的构造函数表示对字符串进行 HTML 编码，并将结果输出发送到文本输出流，参数 s 表示要编码的字符串，output 表示 TextWriter 类的文本输出流。

【范例 14】

下面演示 HtmlEncode()和 HtmlDecode()方法的使用。

（1）创建 BJDefault.aspx 页面，并在页面中添加两个 TextBox 控件和两个 Button 控件，代码如下。

```
<asp:TextBox ID="txtSource" runat="server" TextMode="MultiLine"
Columns="50" Rows="5"></asp:TextBox><br /><br />
<asp:Button ID="btnEncode" runat="server" Text="编 码" OnClick=
"btnEncode Click" /> 
<asp:Button ID="btnDecode" runat="server" Text="解 码" OnClick=
"btnDecode Click" /><br /><br />
<asp:TextBox ID="txtTarget" runat="server" TextMode="MultiLine"
Columns="50" Rows="5"></asp:TextBox>
```

（2）在 BJDefault.aspx.cs 页面添加 Button 控件的 Click 事件代码，以【编码】按钮的事件代码为例，内容如下。

```
protected void btnEncode Click(object sender, EventArgs e) {
    txtTarget.Text = "编码后的内容："+Server.HtmlEncode(txtSource.Text);
}
```

（3）运行窗体页面查看效果，如图 4-17 所示。输入内容后单击【编码】按钮，如图 4-18 所示。

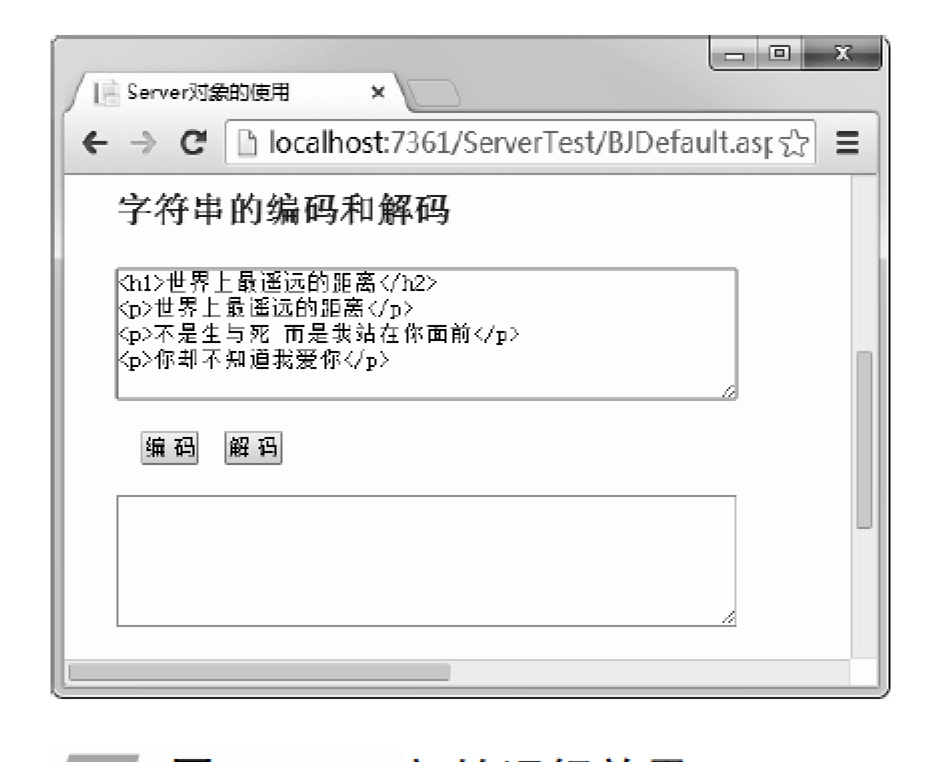


图 4-17 初始运行效果

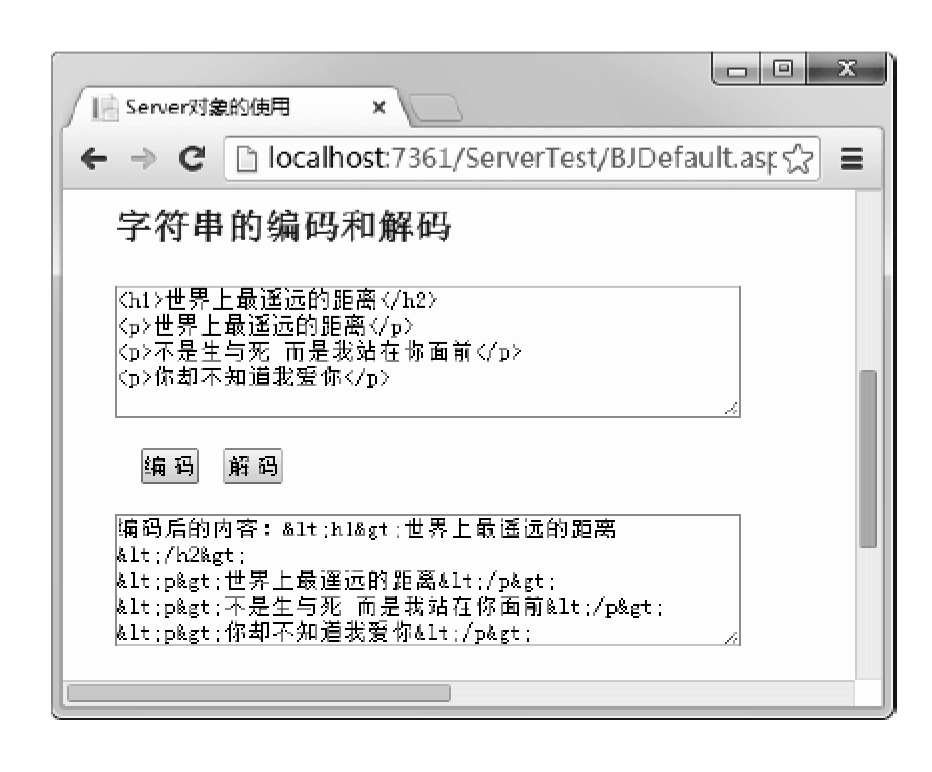


图 4-18 内容编码效果

（4）重新输入内容后单击【解码】按钮进行测试，效果图不再显示。

4.5 实验指导——在窗体页绘制并输出图像数据

本章重点介绍 Page、Response、Request 和 Server 4 个对象，作为 ASP.NET 基本对象之一的 Response 对象不仅可以通过 Write()方法直接在页面上输出字符串数据，还可以

用 BinaryWrite()方法直接显示二进制表示的数据，如图像和图片等。

本节利用 Response 对象在 Web 窗体页中绘制并输出图像数据。实现步骤如下。

（1）在当前的解决方案资源管理器中创建 OutputImage.aspx 窗体页面。

（2）在 OutputImage.aspx.cs 页面后台的 Load 事件中添加代码，首先设置 ContentType 属性，将其指定为"image/jpeg"，以便将整个页面呈现为一幅 JPEG 图像。然后调用 Clear()方法，以确保不会将无关的内容（包括标头）与此响应一同发送。代码如下。

```
protected void Page_Load(object sender, EventArgs e) {
    Response.ContentType = "image/jpeg";
    Response.Clear();
    /* 其他代码 */
}
```

（3）继续在 Load 事件中添加代码，将 BufferOutput 属性的值设置为 true，对响应做出缓冲以便处理完成后发送页面，代码如下。

```
Response.BufferOutput = true;
```

（4）创建 Font 类的实例对象，通过 Font 创建字体风格，代码如下。

```
Font rectangleFont = new Font( "Arial", 10, FontStyle.Bold);
```


（5）声明 int 类型的变量 height 和 width，将 height 变量的值设置为 300，width 变量的值设置为 500。

（6）通过 Random 类创建一个随机数字生成器，并基于该生成器获取随机数的变量 x、a 和 x1，代码如下。

```
Random r = new Random();
int x = r.Next(150);
int a = r.Next(255);
int x1 = r.Next(200);
```

（7）通过 Bitmap 类创建一张位图，并且使用位图对象创建一个 Graphics 对象，代码如下。

```
Bitmap bmp = new Bitmap(width, height, PixelFormat.Format24bppRgb);
Graphics g = Graphics.FromImage(bmp);
g.SmoothingMode = SmoothingMode.AntiAlias;
g.Clear(Color.LightGray);
```

（8）使用步骤（7）创建的 Graphics 对象绘制三个矩形，代码如下。

```
g.DrawRectangle(Pens.Yellow, 1, 1, width - 3, height - 3);
g.DrawRectangle(Pens.Blue, 2, 2, width - 3, height - 3);
g.DrawRectangle(Pens.Black, 0, 0, width, height);
```

（9）通过 Graphics 对象输出一个字符串数据，数据内容是"这是 ASP.NET 例子"，代码如下。

```
g.DrawString("这是ASP.NET例子", rectangleFont, SystemBrushes.WindowText,
new PointF(200, 130));
```

（10）通过 Graphics 对象的 FillRectangle()方法填充矩形，代码如下。

```
g.FillRectangle(new SolidBrush(Color.FromArgb(a, 255, 128, 255)), x, 20,
100, 150);
g.FillRectangle(new LinearGradientBrush(new Point(x, 130), new Point(x1
+ 75, 50 + 30), Color.FromArgb(128, 0, 0, 128), Color.FromArgb(255, 255,
255, 240)), x1, 150, 125, 50);
```

FillRectangle()方法填充由一对坐标、一个宽度和一个高度指定的矩形的内部，基本语法如下：

```
public void FillRectangle(Brush brush, int x, int y, int width, int height);
```

其中，brush 确定填充的矩形框内的背景色；x 和 y 分别表示矩形的左上角顶点位置的 x 坐标和 y 坐标；width 和 height 表示要填充的矩形的宽度和高度。

（11）调用位图对象的 Save()方法把位图保存到响应流中并且把它转换成 JPEG 格式，代码如下。

```
bmp.Save(Response.OutputStream, ImageFormat.Jpeg);
```

（12）调用 Dispose()方法释放掉 Graphics 和位图使用的内存空间，代码如下。

```
g.Dispose();            //释放掉Graphics所使用的内存空间
bmp.Dispose();          //释放掉位图所使用的内存空间
```

（13）调用 Response 对象的 Flush()方法把输出结果发送到客户端，代码如下。

```
Response.Flush();          //把输出结果发送到客户端
```

（14）运行 OutputImage.aspx 窗体页查看效果，由于 FillRectangle()方法填充矩形时的坐标和透明度都是随机产生的，因此，运行页面时可能会看到不同的效果。如图 4-19～图 4-22 所示分别为 Chrome 浏览器、Firefox 浏览器、Opera 浏览器和 IE 浏览器的初始运行效果。



图 4-19　Chrome 浏览器

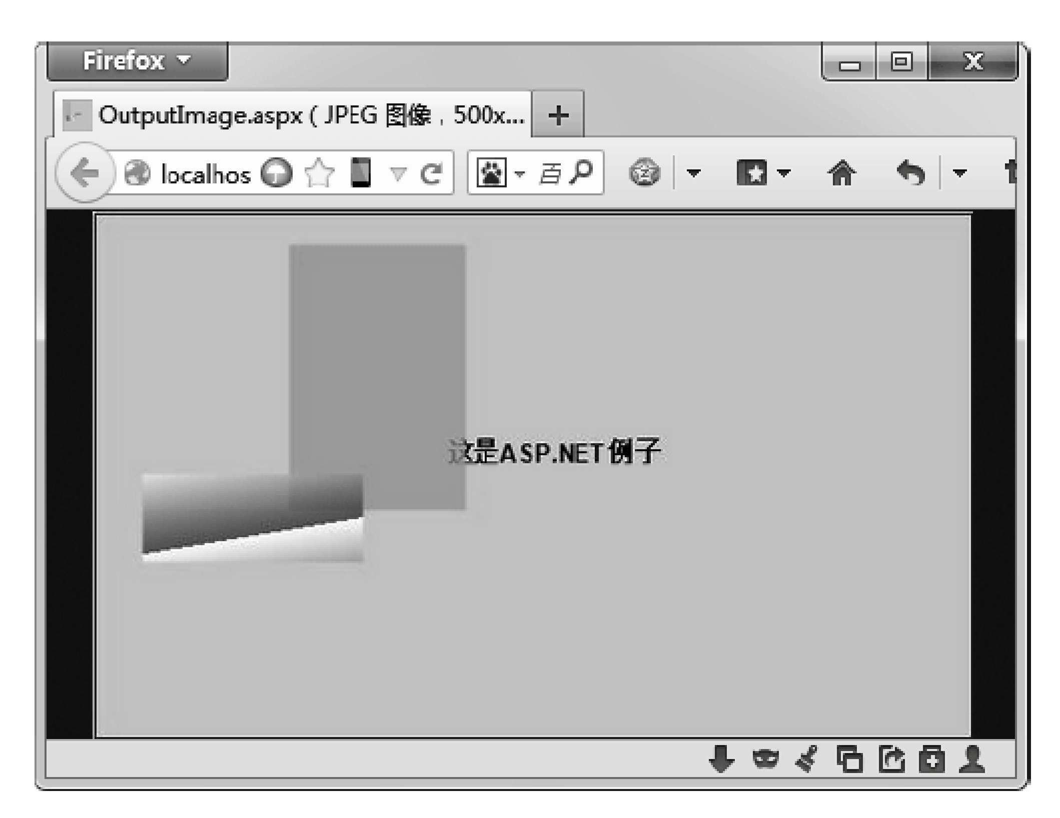


图 4-20　Firefox 浏览器



图 4-21 Opera 浏览器



图 4-22 IE 浏览器

思考与练习

一、填空题

1．Page 对象的__________属性指示页面是否正为响应客户端回发而加载。

2．Response 对象的__________方法表示将客户端重定向到新的 URL。

3．HttpBrowserCapabilities 对象的_______属性获取客户端使用的操作平台的名称。

4．Request 对象的__________属性获取当前请求的虚拟路径。

5．Server 对象的__________属性用于获取服务器的计算机名称。

二、选择题

1．获取客户端表单信息时，不能通过 Request 对象的__________属性获取。

A．Form

B．QueryString

C．Browser

D．Params

2．Response 对象的属性不包括________。

A．ContentType

B．Buffer

C．IsValid

D．Expires

3．Request 对象的_________属性获取远程客户端的 IP 主机地址。

A．UserHostAddress

B．UserHostName

C．UserLanguages

D．UserAgent

4．Server 对象和 Request 对象都包含__________方法。

A．Transfer()

B．MapPath()

C．BinaryRead()

D．CreateObject()

5．Server 对象的__________方法对字符串进行编码，__________方法对字符串进行解码。

A．UrlDecode()，UrlEncode()

B．UrlEncode()，UrlDecode()

C．HtmlDecode()，HtmlEncode()

D．HtmlEncode()，HtmlDecode()

6．Request 对象的 ServerVariables 属性中保存多个信息变量，这些变量不包括__________。

A．ALL_HTTP

B．ALL_RAWS

C．LOCAL_ADDR

D．REMOTE_HOST

7．通过 Page 不能获取__________对象。

A．Application

B．Request

C．Server

D．ViewState

三、简答题

1．请列举 Page 对象的常用属性，并对这些属性说明（至少说出 5 个属性）。

2．Response 对象的常用方法有哪些？这些方法是用来做什么的？

3．Request 对象的常用属性有哪些？这些属性是用来做什么的？

4．实现页面跳转的方法有哪些？

5．Server 对象的常用方法有哪些？这些方法是用来做什么的？

第 5 章　数据保存对象

在实现页面跳转时，开发者可能需要接收从上一个页面中传递过来的数据。例如，实现某个网站的用户登录时，在 Login.aspx 页面登录成功后可跳转到 Default.aspx 页面，后者需要获取从登录页面传递过来的用户名和密码。通过 Response 对象实现页面跳转时，开发者可以通过参数传值，这样会把用户名和密码显示到地址栏中，但是这样会导致密码泄漏，不安全。ASP.NET 专门提供了用于保存数据的对象，本章介绍常用的几个数据保存对象。

本章学习要点：

- ❑ 熟悉 Application 的常用属性
- ❑ 掌握 Application 的常用方法
- ❑ 了解 Application 的常用事件
- ❑ 熟悉 Session 的常用属性和方法
- ❑ 掌握 Session 的创建和使用
- ❑ 了解使用 Cookie 时的注意事件
- ❑ 熟悉 Cookie 的常用属性
- ❑ 掌握 Cookie 的读写和使用
- ❑ 熟悉 ViewState 的注意事项
- ❑ 掌握 ViewState 的读写和使用

5.1 Application 对象

Application 对象是内置的 ASP.NET 对象，表示 ASP.NET 应用程序的示例。Application 对象用来存储变量或对象，以便在网页再次被访问时，所存储的变量或对象的内容还可以被重新调出来使用。也就是说，Application 对象对于同一网站来说是公用的，可以在各个用户间共享。

5.1.1 Application 对象的属性

Application 对象是 HttpApplicationState 类的实例，它包含所有与应用程序相关的方法和集合。HttpApplicationState 类的单个实例，将在客户端第一次从某个特定的 ASP.NET 应用程序虚拟目录中请求任何 URL 资源时创建。对于 Web 服务器上的每个 ASP.NET 应用程序，都要创建一个单独的实例，然后通过内部 Application 对象公开对每个实例的引用。

Application 对象提供了多个属性，通过这些属性可以获取集合中的对象数或者访问

集合中的对象，常用属性如表 5-1 所示。

表 5-1 Application 对象的常用属性

属性名称	说明
Count	获取 HttpApplicationState 集合中的对象数
AllKeys	获取 HttpApplicationState 集合中的访问键
Keys	获取 NameObjectCollectionBase.KeysCollection 实例，该实例包含 NameObjectCollectionBase 实例中的所有键
Contents	获取对 HttpApplicationState 对象的引用

可以通过键名和索引两种方式向 Application 对象中添加数据。

```
Application["keyName"] = objectValue;          //通过键名
Application[0] = objectValue;                  //通过索引
```

【范例 1】

通过键名的形式向 Application 对象中分别添加 totalCount 对象和 comprehensiveScore 对象，最后调用 Application 对象的 Count 属性输出对象数，代码如下。

```
Application["totalCount"] = 12;
Application["comprehensiveScore"] = 98.5;
Response.Write("Application 中保存的对象数：" + Application.Count);
```


5.1.2 Application 对象的方法

Application 对象提供了一系列的方法，通过调用方法也可以实现更改数据的添加、删除和更改。如表 5-2 所示列出了 Application 对象的常用方法，并对这些方法进行说明。

表 5-2 Application 对象的常用方法

方法名称	说明
Add()	将新的对象添加到 HttpApplicationState 集合中
Remove()	从 HttpApplicationState 集合中移除命名对象
RemoveAt()	按索引从集合中移除单个 HttpApplicationState 对象
RemoveAll()	从 HttpApplicationState 集合中移除所有对象
Clear()	从 HttpApplicationState 集合中清除所有对象
GetKey()	通过索引获取 HttpApplicationState 对象名
Get()	通过名称或者索引获取 HttpApplicationState 对象
Set()	更新 HttpApplicationState 集合中的对象值
Lock()	锁定全部的 Application 变量
UnLock()	解除锁定的 Application 变量

1. Add()方法

Application 对象的 Add()方法是指将新的对象添加到 System.Web.HttpApplicationState 集合，语法如下。

```
void HttpApplicationState.Add(string name, object value)
```

其中，name 是指要添加到集合中的对象名，也称为键名；value 是指对象的值。

【范例 2】

在 Web 窗体页的后台 Load 事件中添加代码，调用 Add()方法添加 First、Second 和 Third 对象，然后通过 Application["Fourth"]添加第 4 个对象，最后通过 for 语句遍历集合中的每个对象。代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    Application.Add("First", "程瑶");
    Application.Add("Second", "Lucy");
    Application.Add("Thrid", "飞飞");
    Application["Fourth"] = "木木";
    for (int i = 0; i < Application.Count; i++) {       //遍历 Application
                                                          集合中的对象
        lblMessage.Text += "变量名：" + Application.GetKey(i) + ",变量值：
        " + Application[i]+"<br/>";
    }
}
```

其中，lblMessage 是 Label 控件的 ID 属性值。

运行窗体页面查看效果，如图 5-1 所示。之所以会输出 7 个对象，是因为 Application 是一个全局对象，而 totalCount 和 comprehensiveScore 在范例 1 中已创建，因此这里会将它们显示出来。

虽然 Application["keyName"]和 Application.Add()都可以添加对象，但是它们有很大的区别。当为同一个对象指定属性值时，Application["keyName"]会用新值代替旧值，而 Add()方法依然会添加新的内容。重新运行上面的 BaseTest.aspx 页面，如图 5-2 所示。



图 5-1　Application 添加对象 1



图 5-2　Application 添加对象 2

2. Clear()方法

Clear()方法用于从 HttpApplicationState 集合中清除所有对象。例如，在范例 2 添加对象之前，可以调用 Clear()方法，代码如下。

```
Application.Clear();
```

重新运行或刷新页面，效果如图 5-3 所示。

3．Remove()方法

Remove()方法指定从 HttpApplicationState 集合中移除命名对象，使用时需要传入一个 string 类型的参数，代码如下。

```
void HttpApplicationState.Remove(string name)
```

【范例 3】

在前面范例的基础上添加代码，通过 Remove()方法删除键名为 Second 的对象，然后再通过 for 循环遍历 Application 对象，代码如下。

```
Application.Remove("Second");
for (int i = 0; i < Application.Count; i++) {
                                        //遍历 Application 集合中的对象
    lblMessage.Text += "变量名：" + Application.GetKey(i) + ",变量值：" +
    Application[i] + "<br/>";
}
```

运行页面查看效果，如图 5-4 所示。



图 5-3　Clear()方法的使用



图 5-4　Remove()方法的使用

除了 Remove()方法外，RemoveAll()和 RemoveAt()方法也可以移除指定的对象。RemoveAll()方法移除所有的对象，其效果与 Clear()方法一致。RemoveAt()是按照索引移除指定对象，语法如下。

```
void HttpApplicationState.RemoveAt(int index)
```

4．Set()方法

Set()方法表示更新 HttpApplicationState 集合中的对象值。基本语法如下。

```
void HttpApplicationState.Set(string name, object value)
```

其中，name 指定要更新的对象名；value 指定对象更新之后的值。如果 name 在 Application 中不存在，则会向 HttpApplicationState 中添加一个新对象，对象名为 name，对象值为 value。

【范例 4】

如下代码将 Application 中键名为 Third 的对象的值更改为“abc”。

```
Application.Set("Thrid", "abc");
```

5. GetKey()方法

GetKey()方法表示通过索引获取 HttpApplicationState 对象名。语法如下。

```
void HttpApplicationState.GetKey(int index)
```

在范例 2 通过 for 语句遍历 Application 中的对象时，已经使用到了 GetKey()方法。除了这个方法外，也可以通过以下语法获取指定对象的值。

```
Object value = Application["keyName"];          //根据键名获取
Object value = Application[0];                  //根据索引获取
```

6. Lock()方法和 UnLock()方法

Application 对象使给定应用程序的所有用户之间共享信息，并且在服务器运行期间持久地保存数据。由于多个用户可以共享一个 Application 对象，因此，必须要有 Lock()和 UnLock()方法，以确保多个用户无法同时改变某一属性。Application 对象成员的生命周期止于关闭 IIS 或使用 Clear()方法清除。Lock()和 UnLock()方法的使用语法如下。

```
Application.Lock();
Application["变量"] = "内容";
Application.UnLock();
```


5.1.3 使用 Application 的事件

在 ASP.NET 应用程序中可以包含一个特殊的文件——Global.aspx 文件，也称作 ASP.NET 应用程序文件，它包含用于响应 ASP.NET 或 HTTP 模块引发的应用程序级别事件的代码。Global.asax 文件提供了 7 个事件，其中 5 个应用于 Application 对象。如表 5-3 所示对这些事件进行了说明。

表 5-3 **Application 对象的常用事件**

事件名称	说明
Application_Start	在应用程序启动时激发
Application_BeginRequest	在每个请求开始时激发
Application_AuthenticateRequest	尝试对使用者进行身份验证时激发
Application_Error	在发生错误时激发
Application_End	在应用程序结束时激发

在 VS 2012 中创建基于窗体的应用程序时会自动创建一个 Global.asax 文件。如果没有创建，开发者可以像创建 Web 窗体页那样创建 Global.asax 文件。

统计在线用户的作用是为了网站管理者可以知道当前用户的多少，然后根据用户数量来观察服务器或者程序的性能，从而可以直观地了解到网站的吸引力或者网站程序的效率。下面的范例将 Application 和 Global.asax 文件结合起来，实现在线用户的统计。

【范例 5】

将 Application 和 Global.asax 结合统计在线用户的访问量，步骤如下。

（1）打开当前应用程序中的 Global.asax 文件，如果不存在，需要进行创建。

（2）在 Application_Start 事件中添加代码，初始化统计在线人数的全局变量。如果在线人数为空，则将其初始化为 0，代码如下。

```
protected void Application_Start(object sender, EventArgs e) {
    if (Application["CountOnline"] == null) {
        Application["CountOnline"] = 0;
    }
}
```

（3）在 Session_Start 事件中添加代码，对统计在线人数的全局变量加 1。更改全局变量值之前需要调用 Application 对象的 Lock()方法锁定，更改之后调用 Application 对象的 UnLock()方法解锁。另外，为了方便测试，在更改全局变量之前将 Session 对象的 Timeout 属性值设置为 1，即 Session 的生存时间为 1 min，代码如下。

```
void Session_Start(object sender, EventArgs e) {
    Session.Timeout = 1;
    Application.Lock();
    Application["CountOnline"] = (int)Application["CountOnline"] + 1;
    Application.UnLock();
}
```


（4）在会话对象销毁时要对统计在线人数的全局变量减 1。向 Session_End 事件中添加如下代码。

```
void Session_End(object sender, EventArgs e) {
    Application.Lock();
    Application["CountOnline"] = (int)Application["CountOnline"] - 1;
    Application.UnLock();
}
```

（5）创建 GetVisitPerson.aspx 页面，并在该页面中添加如下代码。

```
<div>当前在线访问人数：<%=Application["CountOnline"] %></div>
```

（6）在多个浏览器中运行 GetVisitPerson.aspx 页面并查看效果，效果图不再显示。

5.2 Session 对象

Session 对象是 HttpSessionState 类的一个实例，该类为当前用户会话提供信息，还提供对可用于存储信息的会话范围的缓存的访问，以及控制如何管理会话的方法。本节简单介绍 Session 对象的属性、方法以及使用。

5.2.1 Session 对象概述

Session 对象的功能和 Application 对象类似，都是用来存储跨网页程序的变量或者对

象，但是 Session 对象和 Application 对象变量有些特性不太一样。各个联机的机器有各自的 Session 对象变量，不同的联机无法互相存取。

Application 对象变量的生命周期终止于停止 IIS 服务，但是 Session 对象变量终止于联机机器离线，也就是当网页使用者关掉浏览器或超过设置 Session 变量对象的有效时间时，Session 对象变量就会消失。

使用 Session 对象存储特定用户会话所需的信息。这样，当用户在应用程序的 Web 页之间跳转时，存储在 Session 对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。

当用户第一次请求指定的应用程序中的“.aspx”文件时，ASP.NET 将生成一个 SessionID，它是由一个复杂算法生成的号码，它唯一标识每个用户会话。在新会话开始时，服务器将 SessionID 作为一个 Cookie 存储在用户的 Web 浏览器中。

在将 SessionID 存储于用户的浏览器之后，即使用户请求了另一个“.aspx”文件，或请求了运行在另一个应用程序中的“.aspx”文件，ASP.NET 仍会重用这个 Cookie 跟踪会话。与此相似，如果用户故意放弃会话或让会话超时，然后再请求另一个“.aspx”文件，那么 ASP.NET 将以同一个 Cookie 开始新的会话。只有当服务器管理员重新启动服务器，或用户重新启动 Web 浏览器时，存储在内存中的 SessionID 设置才被清除，用户将会获得新的 SessionID。

5.2.2 Session 对象的属性

Session 对象和 Application 对象都是属于 Page 对象的成员，因此，可以直接使用。Session 对象的使用方式和 Application 类似，语法如下。

```
Session["变量名"] = "内容";
```

对于每个用户的每次访问，Session 对象都是唯一的，可以通过 Session 对象在页面间共享信息。只要 Session 没有超时，或者 Abandon()方法没有被调用，Session 中的信息就不会丢失。Session 对象不能在用户间共享信息，而 Application 对象可以在不同的用户间共享信息。

Session 对象提供了多个属性，通过这些属性可以设置会话超时的时间，查看当前会话状态模式和唯一标识符等，常用属性如表 5-4 所示。

表 5-4 Session 对象的常用属性

属性名称	说明
CodePage	获取或设置当前会话的字符集标识符
Contents	获取对当前会话状态对象的引用
CookieMode	获取一个值，该值指示是否为无 Cookie 会话配置应用程序
Count	获取会话状态集合中的项数
IsCookieless	获取一个值，该值指示会话 ID 是嵌入在 URL 中还是存储在 HTTPCookie 中
IsNewSession	获取一个值，该值指示会话是否是与当前请求一起创建的
IsReadOnly	获取一个值，该值指示会话是否为只读
Keys	获取存储在会话状态集合中所有值的键的集合

续表

属性名称	说明
LCID	获取或设置当前会话的区域设置标识符（LCID）
Mode	获取当前会话状态模式
SessionID	获取会话的唯一标识符
Timeout	获取并设置在会话状态提供程序终止会话之前各请求之间所允许的时间（以 min 为单位）。默认为 20 min

【范例 6】

调用表 5-4 中的 Mode 属性、SessionID 属性和 Count 属性获取会话模式、当前会话 SessionID 和会话集合中的项数，代码如下。

```
Response.Write("当前会话模式：" + Session.Mode + "<br/>");
Response.Write("当前会话 SessionID：" + Session.SessionID + "<br/>");
Response.Write("会话集合中的项数：" + Session.Count + "<br/>");
Response.Write("会话有效时间：" + Session.Timeout);
```

5.2.3 Session 对象的方法

Session 对象提供了一系列操作会话的方法，通过这些方法可以添加新会话，也可以删除指定的会话。如表 5-5 所示对 Session 对象的常用方法进行说明。

表 5-5 Session 对象的常用方法

方法名称	说明
Abandon()	取消当前会话
Add()	向会话状态集合添加一个新项
Clear()	从会话状态集合中移除所有的键和值
Remove()	删除会话状态集合中的项
RemoveAll()	从会话状态集合中移除所有的键和值
RemoveAt()	删除会话状态集合中指定索引处的项
CopyTo()	将会话状态值的集合复制到一维数组中（从数组的指定索引处开始）

1. Add()方法

Session 对象的 Add()方法表示向会话状态集合中添加一个新项，语法如下。

```
void HttpSessionState.Add(string name, object value);
```

其中，name 表示要添加到会话状态集合的项的名称；value 表示要添加到会话状态集合的项的值。

【范例 7】

分别通过 Add()方法和赋值的方式向会话中添加对象，然后通过 Session.Count 输出集合中的对象数，代码如下。

```
Session.Add("loginPass", "234523");
Session["tranPass"] = "909873";
```

```
Response.Write("会话集合中的对象数：" + Session.Count);
```

2. CopyTo()方法

CopyTo()方法是指将会话状态值的集合复制到一维数组中（从数组的指定索引处开始），语法如下。

```
void HttpSessionState.CopyTo(Array array, int index)
```

其中，array 是一个数组，它接收会话值；index 指定 array 中从零开始的索引，在此处开始复制。

【范例 8】

首先通过 Session 存储多个对象，接着声明 strArray 一维数组，并遍历数组中的值。然后调用 CopyTo()方法将会话状态值复制到数组，最后遍历 strArray 数组中的元素，代码如下。

```
Session["pass1"] = "666666";            //为 session 变量赋值
Session["pass2"] = "888888";
Session["pass3"] = "556985";
String[] strArray = new String[] { "1", "2", "3", "4" };
                                        //建立字符串数组
Response.Write("原数组如下：</br>");
foreach (String str in strArray) {      //数组遍历
    Response.Write(str + "</br>");
}
Session.CopyTo(strArray, 0);            //将 session 会话状态值复制到数组
Response.Write("新数组如下：</br>");
foreach (string str in strArray) {
    Response.Write(str + "</br>");
}
```

提 示

Global.asax 文件提供的 7 个事件中，Session_Start 和 Session_End 事件与 Session 对象有关。Session_Start 创建一个会话，在 Session 对象创建时触发。Session_End 结束一个会话，在 Session 对象销毁时触发。

5.3 实验指导——用户的安全登录和退出

一般情况下，开发者可以使用 Session 保存一些比较敏感的数据，例如登录账户名和登录密码。本节模拟实现用户的安全登录和退出功能，步骤如下。

（1）创建名称为 UserInfo 的实体类，该类包含 loginName 和 loginPass 两个字段，通过属性对这两个字段进行封装。代码如下。

```
public class UserInfo
{
```

```
        public UserInfo() { }
        private string loginName;
        public string LoginName {
            get { return loginName; }
            set { loginName = value; }
        }
        private string loginPass;
        public string LoginPass {
            get { return loginPass; }
            set { loginPass = value; }
        }
    }
```

（2）创建 Login.aspx 窗体页，向该页面的表单中添加【登录名】输入框、【登录密码】框和【登录】按钮。代码如下。

```
    <table align="center" height="100" width="100%">
        <tr><td align="right">登录名：</td><td><asp:TextBox ID="txtLoginName"
        runat="server"></asp:TextBox></td></tr>
        <tr><td align="right">登录密码：</td><td><asp:TextBox ID="txtLoginPass"
        TextMode="Password" runat="server"></asp:TextBox></td></tr>
        <tr><td></td><td><asp:Button ID="btnLogin" runat="server" Text="登 录
        " OnClick="btnLogin_Click" /></td></tr>
    </table>
```

（3）在后台页面为步骤（2）中的 Button 控件添加 Click 事件代码，首先获取用户输入的登录名和登录密码，接着将登录名和密码赋值为 UserInfo 的实例对象，然后通过 Session 保存 user 对象，最后通过 Response 对象的 Redirect()方法跳转页面。代码如下。

```
    UserInfo user = new UserInfo();
    protected void btnLogin_Click(object sender, EventArgs e) {
        string name = txtLoginName.Text;     //获取登录名
        string pass = txtLoginPass.Text;     //获取登录密码
        user.LoginName = name;
        user.LoginPass = pass;
        Session["user"] = user;
        Response.Redirect("Default.aspx");
    }
```

（4）创建 Default.aspx 页面，向该页面添加一个 Label 控件和一个 LinkButton 控件，前者用于显示登录名和登录密码，后者指定一个【退出】按钮。代码如下。

```
    <asp:Label ID="lblMessage" runat="server" Font-Size="Larger" ForeColor="Red">
    </asp:Label>
    <asp:LinkButton ID="lbtnExit" runat="server" Text="退 出" OnClick=
    "lbtnExit_Click"></asp:LinkButton>
```

（5）在后台页面为步骤（4）中的【退出】按钮添加 Click 事件代码。

```
    protected void lbtnExit_Click(object sender, EventArgs e) {
```

```
        Session.Abandon();              //取消当前会话
        Session.Remove("user");         //删除 user 会话对象
        if (Session["user"] == null) {
            Response.Write("<script>alert('您已经成功退出，请到登录页进行登录
            ');window.location.href='Login.aspx';</script>");
        }
    }
```

在上述代码中，首先调用 Session 对象的 Abandon()方法取消当前会话，然后调用 Remove()方法删除 user 会话对象，最后通过 if 语句判断 Session 中保存的 user 对象是否为空，如果为空，则调用 Response 对象的 Write()方法弹出提示并跳转页面。

（6）在 Defalut.aspx 窗体页面的后台 Load 事件中添加代码。在该事件中判断获取到的 user 对象是否为空，如果为空则跳转页面；否则通过 as 将取出的内容转换为 UserInfo 对象，再将获取到的内容显示到 Label 控件中。代码如下。

```
    protected void Page_Load(object sender, EventArgs e) {
        if (Session["user"] == null) {
            Response.Write("<script>alert('您可能没有登录，找不到您的信息，单击按
            钮跳转到登录页面。');window.location.href='Login.aspx';</script>");
        } else {
            UserInfo user = Session["user"] as UserInfo;
            lblMessage.Text = "您的登录名是：" + user.LoginName + "，  登录密码是：
            " + user.LoginPass;
        }
    }
```


（7）直接运行 Default.aspx 页面查看效果，由于首次运行没有获取到 Session 中保存的 user 对象，因此会弹出对话框提示，如图 5-5 所示。单击【确定】按钮跳转到登录页面，如图 5-6 所示。

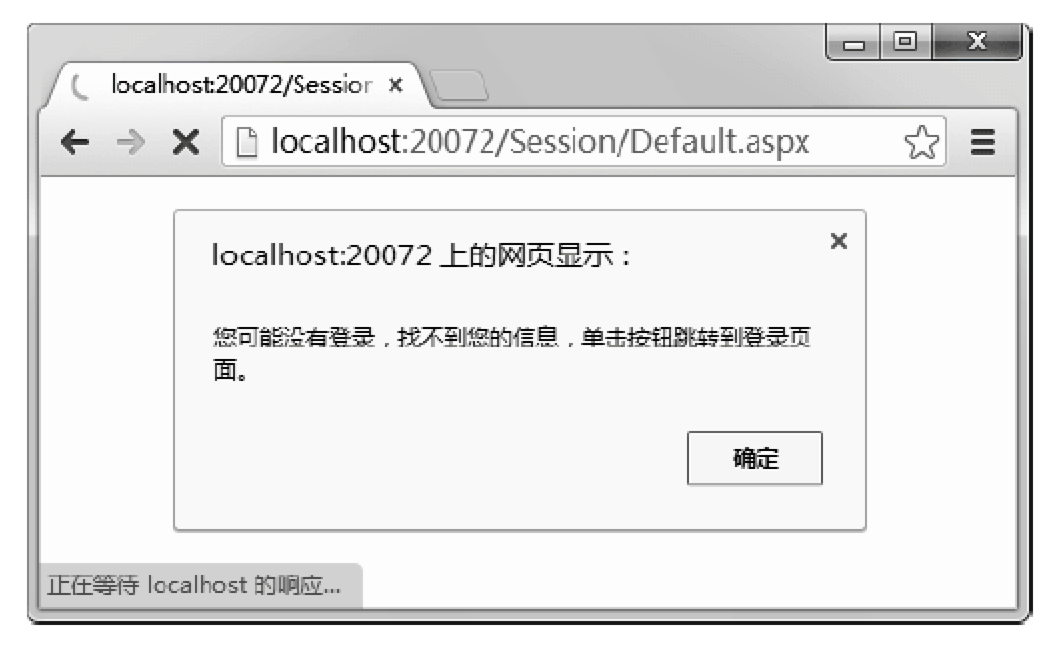


图 5-5　提示登录



图 5-6　登录页面

（8）在如图 5-6 所示界面中输入登录名和登录密码，然后单击【登录】按钮，如图 5-7 所示。

（9）打开一个新的窗口，直接在地址栏中输入 Default.aspx 页面的地址进行访问，如图 5-8 所示。这时可以发现，程序已经检测到 Session 中的 user 对象，而不再提示用户重新登录。

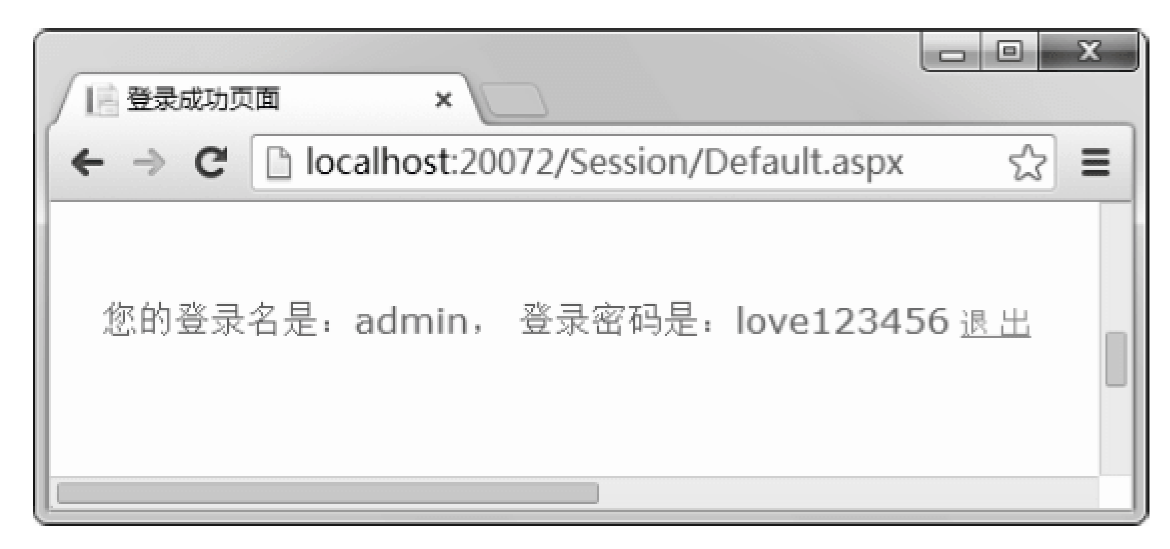


图 5-7 用户登录成功时的页面

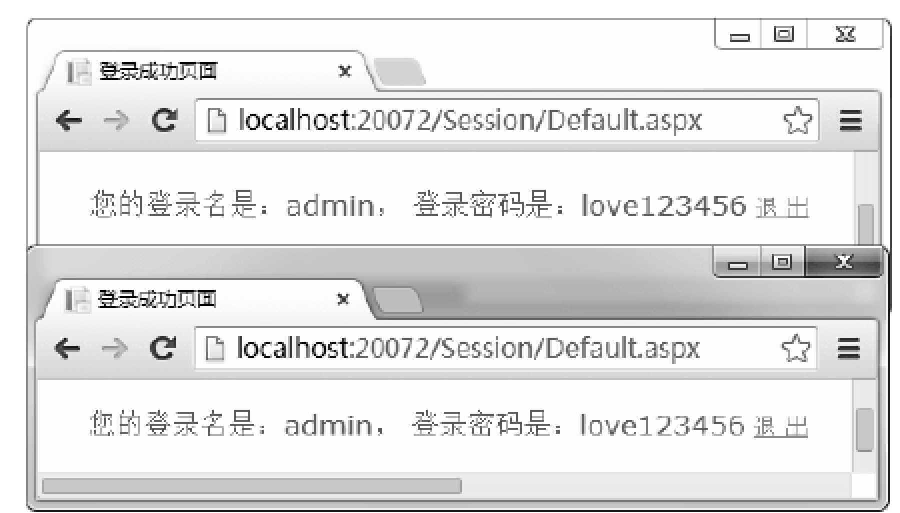


图 5-8 打开另一个窗口

（10）单击图 5-8 中的【退出】按钮，然后直接刷新或重新访问 Default.aspx 页面，效果图不再显示。

5.4 Cookie 对象

Session 对象用于保存安全性较高的数据，如果有些数据不是特别重要（例如用户地址），这时再用 Session 保存会使服务器内存资源的占用量大，过多的存储会导致服务器内存资源耗尽。

ASP.NET 中提供了多个内置对象，提到 Session 对象，就不得不提到另一个保存数据的对象，即 Cookie 对象。

5.4.1 Cookie 对象概述

Cookie 对象提供一种在 Web 应用程序中存储用户特定信息的方法。实际上，Cookie 就是保存在客户机硬盘上的一个文件，它允许 Web 应用程序在用户的计算机上保存并取回信息，信息的片段以键/值对的形式存储。

1. 写入 Cookie

由于 Cookie 不是 Page 类的子类，因此，在使用方法中与 Session 和 Application 等对象有所不同。Cookie 对象分别属于 Request 对象和 Response 对象，每一个 Cookie 变量都是被 Cookies 对象所管理，它的对象类型名称是 HttpCookieCollection。

如果要写入或者存储一个 Cookie 变量，需要用 Response 对象的 Cookies 集合。使用方法如下。

```
Response.Cookies[Cookie 的索引值].Value = "变量值";
Response.Cookies["Cookie 的名称"].Value = "变量值";
```

调用 Response.Cookies.Add()方法也可以将指定的 Cookie 添加到 Cookie 集合中，语法如下。

```
HttpCookie hcCookie = new HttpCookie("Cookie 的名称","值");
                                                        //第二种创建方法
Response.Cookies.Add(hcCookie);
```

2．读取 Cookie

读取 Cookie 时需要借助于 Request 对象的 Cookies 集合，并将指定的 Cookie 传回，语法如下。

```
变量名 = Request.Cookies["Cookie 的名称"].Value;
变量名 = Request.Cookies[Cookie 的索引值].Value;
```

除了上述方法外，还可以通过 Request.Cookies.GetKey()方法进行读取，语法如下。

```
string 变量名 = Request.Cookies.GetKey(Cookie 的索引值);
```

3．Cookie 的注意事项

Cookie 通常用于保存非敏感的用户信息，如居住地址、上次登录时间和上次登录 IP 地址等。开发者在使用 Cookie 时需要注意以下几点。

（1）并非所有的浏览器都提供对 Cookie 的支持。

（2）Cookie 对象的有效时间默认为保存至用户关闭浏览器，这时 Cookie 对象将不会被保存至用户硬盘。

（3）Cookie 对象数据信息是以明文文本的形式保存在客户端的计算机中，因此，不能保存敏感的、未加密的数据，以确保网站的安全性。

（4）Cookie 对象存储数据量有限，大多数浏览器支持最大容量为 4KB，因此，不能保存数据量大的数据。

4．Cookie 与 Session 和 Application 的区别

Cookie 与 Session 和 Application 对象很类似，也是一种集合对象，都用来保存数据。但是它们也存在着区别，如下从存储位置、生命周期、应用范围以及信息量大小等方面进行说明。

1）存储位置不同

这是 Cookie 与 Session 和 Application 对象的最大不同点。Cookie 以档案的形式将数据存放在客户端的磁盘上，而 Application 和 Session 对象是将数据存放于服务器端。

2）生命周期不同

Cookie 的生命周期可以一直存在，也可以终止于所设置的时间。Application 对象的生命周期终止于 IIS 关闭时。Session 对象的生命周期终止于设置的时间或使用者离线。

3）应用范围

Cookie 和 Session 都适用于单个用户，而 Application 则适用于所有的用户，即整个应用程序。

4）信息量大小

Cookie 通常用于存储小量的、简单的、对安全要求不严格的数据。Session 保存小量的、简单的、比较安全的数据，但是它的效率不高。Application 可用于保存任意大小的数据。

5.4.2 Cookie 对象的属性

Cookie 与 Session 和 Application 对象类似，都是用来保存相关数据信息。5.4.1 节已经提到：Cookie 和其他对象最大的不同在于存储位置，即 Cookie 将信息保存在客户端，Session 和 Application 是保存在服务器端。也就是说，无论何时用户连接到服务器，Web 应用程序都可以访问 Cookie 信息。这样，既方便用户的使用，也方便网站对用户的管理。

Cookie 对象实际上是 HttpCookie 类的一个实例，每一个 Cookie 对象都属于 Cookies 集合。Cookie 中包含一系列的属性，常用属性如表 5-6 所示。

表 5-6 **Cookie 对象的常用属性**

属性名称	说明
Name	获取或设置 Cookie 的名称
Value	获取或设置单个 Cookie 值
Values	获取单个 Cookie 对象所包含的键值对的集合
Path	获取或设置与当前 Cookie 一起传输的虚拟路径
Expires	获取或设置 Cookie 的过期日期和时间

【范例 9】

在创建的窗体页中添加 Label 控件，该控件用于显示 Cookie 中的内容。在窗体页面后台的 Load 事件中添加代码，首先增加两个 Cookie 对象，然后使用 for 循环将集合中的 Cookie 对象输出，代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    Response.Cookies["Cookie1"].Value = "Microsoft VisualStudio .NET";
    Response.Cookies["Cookie2"].Value = "ASP.NET";
    lblMessage.Text = "第一种方式：<br/>";
    for (int i = 0; i < Request.Cookies.Count; i++) {
        lblMessage.Text += "变量名称：" + Request.Cookies[i].Name + ",变量
        值：" + Request.Cookies[i].Value+"<br/>";
    }
    lblMessage.Text += "第二种方式：<br/>";
    for (int i = 0; i < Request.Cookies.Count; i++) {
        lblMessage.Text += "变量名称：" + Request.Cookies.GetKey(i) + ",
        变量值：" + Request.Cookies.Get(i).Value+"<br/>";
    }
    Response.Cookies.Clear();
}
```

在上述代码中，通过两种方法获取集合中的对象，除了 Request.Cookies[i].Name 和 Request.Cookies.GetKey(i)可以获取对象名称外，还可以通过 Request.Cookies.Get(i).Name 获取对象名称。

运行本次范例的窗体页查看效果，如图 5-9 所示。

在 ASP.NET 中 Cookie 有两种类型：会话 Cookie（默认类型）和持久性 Cookie。会

话 Cookie 是临时性的，一旦会话状态结束它将不复存在；持久性 Cookie 具有指定的过期日期，过期之前 Cookie 在用户的计算机上以文本文件的形式存储。如果要设置 Cookie 的过期日期，可以通过 Expire 属性进行设置。如下代码设置 Cookie1 对象的过期日期。



图 5-9　读取 Cookie 对象的内容

```
Response.Cookies["Cookie1"].Expires = DateTime.Now.AddMinutes(1);
```

5.5　实验指导——Cookie 对象实现免登录

经常上网的开发者会知道，用户在某个系统或网站的登录页实现登录功能时，通常有一个复选框提示用户是否在某一段时间内免登录，如三天内免登录、一周内免登录。本节模拟实现用户登录功能，通过 Cookie 保存过期时间，在两分钟内实现免登录。步骤如下。

（1）创建 Login.aspx 窗体页，在页面的表单元素中添加【登录名】输入框、【登录密码】输入框、【两分钟内免登录】复选框、【登录】按钮以及显示信息的 Label 控件。代码如下。

```
<table align="center" height="150" width="50%">
    <tr><td align="right" width="30%">登录名：</td><td><asp:TextBox
    ID="txtLoginName" runat="server"></asp:TextBox></td></tr>
    <tr><td align="right">登录密码：</td><td><asp:TextBox ID=
    "txtLoginPass" runat="server"></asp:TextBox></td></tr>
    <tr><td></td><td><asp:CheckBox ID="cbCheck" runat="server" Text="两
    分钟内免登录" /></td></tr>
    <tr><td></td><td><asp:Button ID="btnLogin" runat="server" Text="登 录
    " OnClick="btnLogin_Click" /></td></tr>
    <tr><td></td><td><asp:Label ID="lblMessage" runat="server"
    ForeColor="Red"></asp:Label></td></tr>
</table>
```

（2）在后台页面为步骤（1）中的 Button 控件添加 Click 事件代码。首先判断用户输入的登录名是否等于 admin，如果是则分别保存名称为 CookieTime 和 Login 的 Cookie 对象。然后判断是否选中复选框，如果选中则设置 Cookie 对象的 Expires 属性。最后调用 Response 对象的 Redirect()方法跳转页面。代码如下。

```
protected void btnLogin_Click(object sender, EventArgs e) {
    if (txtLoginName.Text == "admin") {     //登录名必须是admin，密码随意
        Response.Cookies["CookieTime"].Value = txtLoginName.Text;
        Response.Cookies["LoginTime"].Value = DateTime.Now.ToString
        ("yyyy-MM-dd hh:mm:ss");
        if (cbCheck.Checked) {          //确定在两分钟内免登录
            Response.Cookies["CookieTime"].Expires = DateTime.Now.
            AddMinutes(1);
        }
        Response.Cookies["LoginTime"].Expires = DateTime.MaxValue;
        Response.Redirect("~/Default.aspx");
    } else {
        lblMessage.Text = "用户名错误";
    }
}
```

（3）打开当前应用程序根目录下的 Defalut.aspx 页面，在前台页面添加 Label 控件和 LinkButton 控件。代码如下。

```
<asp:Label ID="lblMessage" runat="server" ForeColor="Red"></asp:Label>
<asp:LinkButton ID="lbExit" runat="server" Text="退 出" OnClick=
"lbExit_Click"></asp:LinkButton>
```

（4）在后台中获取名称为 CookieTime 和 LoginTime 的 Cookie 对象，并判断它们的值是否为空，如果为空弹出提示并跳转页面，否则获取 LoginTime 中保存的值并输出到页面。代码如下。

```
protected void Page_Load(object sender, EventArgs e) {
    if (Request.Cookies["CookieTime"] == null || Request.Cookies
    ["LoginTime"] == null) {
        Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "",
        "<script>alert('不能查看页面内容，您可能没有登录，单击按钮确认登录
        ');window.location.href='Cookie/Login.aspx';</script>");
    } else {
        string loginTime = Request.Cookies["LoginTime"].Value;
        lblMessage.Text = "当前时间：" + DateTime.Now.ToString("yyyy-MM-dd
        hh:mm:ss") + "，上次登录时间是：" + loginTime;
    }
}
```

（5）为 Default.aspx 页面中【退出】按钮添加 Click 事件，重新设置 Cookie 对象的过期时间。代码如下。

```
protected void lbExit_Click(object sender, EventArgs e) {
    HttpCookie cookie = Request.Cookies["CookieTime"];
    if (cookie != null) {
        cookie.Expires = DateTime.Now.AddMinutes(-2);
        Response.Redirect("Cookie/Login.aspx");
        Response.Cookies.Set(cookie);
```

```
        }
    }
```

（6）直接运行 Default.aspx 页面查看效果，如图 5-10 所示。单击【确定】按钮跳转到登录页面，如图 5-11 所示。

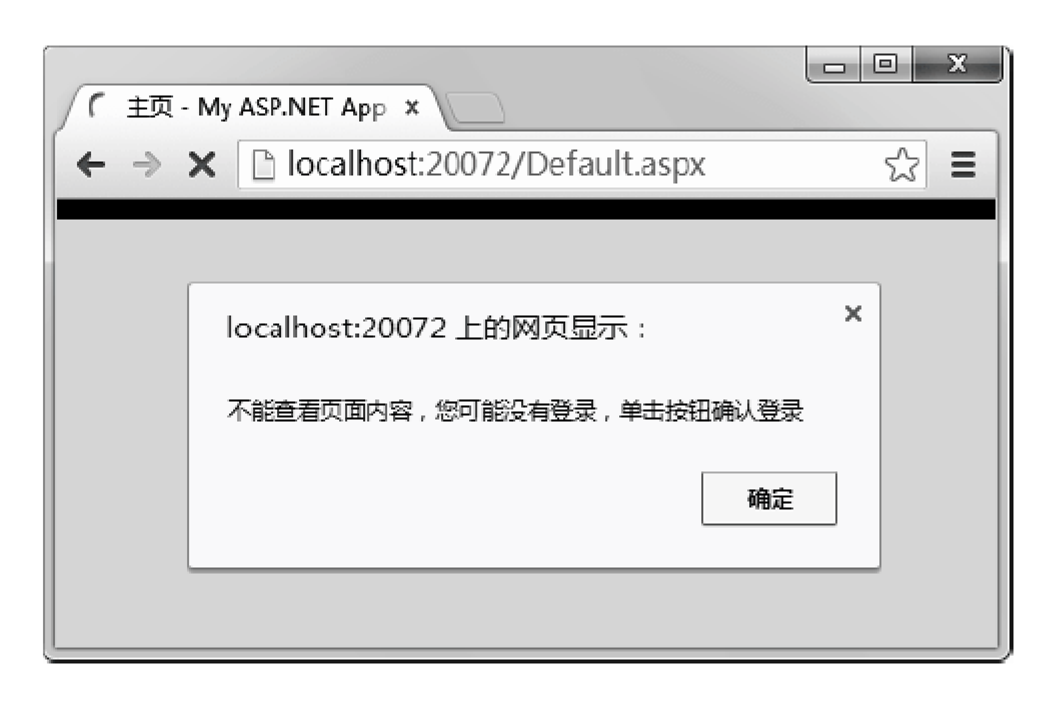


图 5-10 提示内容



图 5-11 登录页面

（7）在如图 5-11 所示界面中输入登录名和登录密码，登录名必须为 admin，否则会提示出错，如图 5-12 所示。重新在登录页面输入登录名和密码，并且选中图中的复选框，成功时的效果如图 5-13 所示。



图 5-12 登录错误



图 5-13 登录成功

（8）刷新页面或者重新打开一个新的窗口查看效果，效果图不再显示。

提 示

本节实验指导模拟实现用户的免登录功能，在一个完整的功能强大的系统中，仅依靠本节的代码是不行的，它们存在着漏洞。例如，同一用户 admin 在不同的浏览器登录，这样可视为重复登录。感兴趣的读者可以重新完善本节的代码，这里不再显示具体的实现过程。

5.6 ViewState 对象

ViewState 对象是 ASP.NET 中用来保存 Web 控件回传时状态值的一种机制。简单来说，ViewState 用于维护页面的 UI 状态。它是类 Control 中的一个域，其他所有控件通过

继承 Control 来获得 ViewState 功能。ViewState 的类型是 system.Web.UI.StateBag，它是一个名称/值的对象集合。

5.6.1 ViewState 对象概述

ViewState 对象并不神秘，如果要使用这个对象，则在“.aspx”页面中必须有一个服务器端窗体标记，即<form runat="server">。窗体字段是必需的，这样包含 ViewState 信息的隐藏字段才能回传给服务器。而且，form 窗体还必须是服务器端的窗体，这样在服务器上执行页面时，ASP.NET 页面框架才能添加隐藏的字段。实际上，在为窗体页面的 form 控件设置 runat="server"时，这个 form 会被附加一个隐藏的_VIEWSTATE 属性。_VIEWSTATE 属性中存放了所有控件的 ViewState 中的状态值。

当请求某个页面时，ASP.NET 把所有控件的状态序列化成一个字符串，然后作为窗体的隐藏属性送到客户端。当客户端把页面回传时，ASP.NET 分析回传的窗体属性，并赋给控件对应的值。这些全部都是由 ASP.NET 负责的，因此对用户来说，这些操作全都是透明的。

ViewState 可在控件、页、程序和全局配置中设置，其优先级别是控件>页面>程序>全局配置。开发者在使用 ViewState 时需要注意以下几点。

（1）ViewState 的索引是大小写敏感的。

（2）ViewState 信息不是网站共享的，因此不能够跨页面。

（3）保存在 ViewState 中的对象必须是可流化或者定义了 TypeConverter。

（4）只有当页面回传（Server.Transfer()）自身时，ViewState 才是持续的，页面跳转（Response.Redirect()）会使 ViewState 中的数据丢失。

（5）当禁止一个程序的 ViewState 时，这个程序的所有页面的 ViewState 也被禁止了。

（6）并不是所有的应用程序都需要保存控件状态信息，通过在页面的@Page 指令中添加 EnableViewState 属性的值为 false 可以禁止整个页面的 ViewState。

（7）TextBox 控件的 TextMode 属性的值设置为 Password 时，其状态不会被保存在 ViewState 中。

> **注 意**
>
> **并不是所有的控件都可以禁止** ViewState，**如** TextBox、CheckBox、CheckBoxList **和** RadioButtonList。**这是因为它们的状态是通过** IsPostBackEventHandler **和** IPostBackDataHandler **接口处理，而不是** ViewState **的机制，因此，这些控件设置** EnableViewState **时没有效果。**

5.6.2 使用 ViewState 对象

由于 ViewState 不使用服务器资源、不会超时，并且适用于任何浏览器，因此，它为跨回传跟踪控件的状态提供了一条神奇的途径。使用 ViewState 读取和写入数据的方式与 Session 和 Application 对象相似，格式如下。

```
ViewState["key"]="value";                    //存放信息
string key=ViewState["key"].ToString();     //读取信息
```

如果要在 Web 窗体页面显示一个项目列表，而每个用户需要根据不同的列表排序。项目列表是静态的，因此，可以将这些页面绑定到相同的缓存数据集，而排序顺序只是用户特定的 UI 状态的一部分，这时，可以使用 ViewState 存储这种类型的值。下面通过范例 10 进行说明。

【范例 10】

本范例利用 GridView 控件显示 XML 文档中的数据列表。使用 ViewState 存储一列静态数据的当前排序顺序，单击列表中的标题链接时，可按字段排序数据，再次单击链接，则按相反的顺序排序。步骤如下。

（1）创建名称为 TestData 的 XML 文档，它包含一系列的图书。部分内容如下。

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Table>
    <pub id>0736</pub id>
    <pub name>New Moon Books</pub name>
    <city>Boston</city>
    <state>MA</state>
    <country>USA</country>
  </Table>
  <!-- 省略其他内容 -->
</NewDataSet>
```

（2）在创建的 Web 窗体页中添加 GridView 控件，代码如下。

```
<asp:DataGrid  ID="DataGrid1"  runat="server"  OnSortCommand="SortGrid"
BorderStyle="None" BorderWidth="1px" BorderColor="#CCCCCC" BackColor=
"White" CellPadding="5" AllowSorting="True" Width="80%">
    <HeaderStyle Font-Bold="True" ForeColor="White" BackColor=
"#006699"></HeaderStyle>
    <ItemStyle ForeColor="Blue" BackColor="#569c66" />
</asp:DataGrid>
```

（3）打开窗体页面的后台添加代码，在 ViewState 中跟踪 SortField 属性，代码如下。

```
public string SortField {
    get {
        object o = ViewState["SortField"];
        if (o == null) {
            return String.Empty;
        }
        return (string)o;
    }
    set {
        if (value == SortField) {
            SortAscending = !SortAscending;// 与当前排序文件相同，切换排序方向
```

```
            }
            ViewState["SortField"] = value;
        }
    }
```

（4）继续添加代码，在 ViewState 中跟踪 SortAscending 属性，代码如下。

```
    public bool SortAscending {
        get {
            object o = ViewState["SortAscending"];
            if (o == null) {
                return true;
            }
            return (bool)o;
        }
        set {
            ViewState["SortAscending"] = value;
        }
    }
```

（5）为页面的 Load 事件添加代码，页面首次加载时调用自定义的 BindGrid()方法，代码如下。

```
    protected void Page Load(object sender, EventArgs e) {
        if (!Page.IsPostBack) {
            BindGrid();
        }
    }
```

（6）BindGrid()方法读取 XML 文档中的数据，并且将数据绑定到 GridView 控件，代码如下。

```
    public void BindGrid() {
        DataSet ds = new DataSet();         //获取数据
        ds.ReadXml(Server.MapPath("TestData.xml"));
        DataView dv = new DataView(ds.Tables[0]);
        dv.Sort = SortField;               //应用排序过滤器和方向
        if (!SortAscending) {
            dv.Sort += " DESC";
        }
        DataGrid1.DataSource = dv;         //绑定 GridView 控件
        DataGrid1.DataBind();
    }
```

（7）为 GridView 控件的 SortCommand 事件添加代码，在该事件中将当前页的索引设置为 0，然后设置 SortField 属性的值，最后调用 BindGrid()方法重新实现绑定。代码如下。

```
    protected void DataGrid1 SortCommand(object source,
        DataGridSortCommandEventArgs e) {
```

```
        DataGrid1.CurrentPageIndex = 0;
        SortField = e.SortExpression;
        BindGrid();
    }
```

（8）运行页面查看效果，如图 5-14 所示。



pub_id	pub_name	city	state	country
0736	New Moon Books	Boston	MA	USA
0517	Binnet & Hardley	Washington	DC	USA
4122	Algodata Infosystems	Berkeley	CA	USA
1025	Five Lakes Publishing	Chicago	IL	USA
1756	Ramona Publishers	Dallas	TX	USA
9901	GGG&G	Muenchen		Germany
4655	Scootney Books	New York	NY	USA
3827	Lucerne Publishing	Paris		France

图 5-14　页面初始运行效果

（9）单击图 5-14 中的标题实现排序功能，如图 5-15 所示为根据 city 进行排序时的效果。



pub_id	pub_name	city	state	country
4122	Algodata Infosystems	Berkeley	CA	USA
0736	New Moon Books	Boston	MA	USA
1025	Five Lakes Publishing	Chicago	IL	USA
1756	Ramona Publishers	Dallas	TX	USA
9901	GGG&G	Muenchen		Germany
4655	Scootney Books	New York	NY	USA
3827	Lucerne Publishing	Paris		France
0517	Binnet & Hardley	Washington	DC	USA

图 5-15　根据 city 排序效果

思考与练习

一、填空题

1．__________是一个全局对象，它是 HttpApplicationState 类的实例。

2．Session 对象的__________属性用于设置超时时间。

3．Cookie 对象的__________属性用于获取 Cookie 的名称。

4．下面代码的横线处应该填写__________。

```
    HttpCookie    ckName    =    new
HttpCookie("loginName","许飞");
    Response.____________.Add(ckName)
```

5．__________是 ASP.NET 中用来保存 Web 控件回传时状态值的一种机制。

二、选择题

1．Application 对象的__________方法表示从 HttpApplicationState 集合中移除命名对象。

A．Remove()

B．RemoveAt()

C．RemoveAll()

D．Clear()

2．Cookie 对象的__________属性可以设置 Cookie 的过期时间和日期。

A．Value

B．Path

C．Expires

D．HasKeys

3．关于 Session、Cookie 和 Application 对象，下面说法错误的是__________。

A．Session 和 Cookie 适用于单个用户，而 Application 适用于所有的用户

B．Session 和 Cookie 用于存储小量的、简单的、对安全要求不严格的数据，而 Application 用于存储大量的、复杂的、对安全要求较高的数据

C．Session 和 Application 将数据存储在服务器端，而 Cookie 以档案的形式将数据存储在客户端的磁盘上

D．默认情况下，浏览器关闭时 Cookie 的生命周期就已经结束，但是可以通过 Expires 属性设置过期时间

4．关于 ViewState 对象，下面说法正确的是__________。

A．ViewState 的索引是大小写敏感的

B．保存在 ViewState 中的对象必须是可流化或者定义了 TypeConverter

C．ViewState 信息不是网站共享的，但是能够跨页面实现

D．TextBox 控件的 TextMode 属性值设置为 Password 时，状态不会被保存在 ViewState 中

三、简答题

1．说出 Application、Session 和 Cookie 之间的异同点。

2．Session 对象的常用属性和方法有哪些？请对这些属性和方法进行说明。

3．如何创建和读取 Cookie 对象？

4．ViewState 是什么？使用时应该注意哪些事项？

第 6 章　站点导航控件

站点导航相当于为网站定义一个目录结构，而导航控件将这个目录结构展示给用户，为用户提供页面链接，使用户能够轻松地找到需要进入的页面。

站点导航控件可以和站点地图结合使用，也可以和 XML 文档结合使用。本章详细介绍如何使用站点地图和 XML 文档定义页面逻辑结构，以及站点导航控件的使用。

本章学习要点：

- 了解导航控件的一般步骤
- 掌握站点地图的使用
- 掌握 SiteMapPath 控件的使用
- 掌握 TreeView 控件的使用
- 掌握 TreeNode 对象的创建
- 理解 TreeView 控件的样式控制
- 掌握 Menu 控件的使用

6.1 导航

一个网站系统通常包含众多页面，导航为网站的用户使用网页起到了引导作用，能够使用户方便快捷地找到自己需要的数据。本节介绍站点导航的基础和站点地图的使用。

6.1.1 导航控件

网站通常都有着一定的页面结构，但这个结构并不是存放在导航控件中，而是存放在站点地图或 XML 文档中，导航的实现通常需要以下几步。

（1）将网站中页面的逻辑结构和链接存储在站点地图中。站点地图是一个名为 Web.sitemap 的 XML 文件，描述站点的逻辑结构。

（2）使用 SiteMapDataSource 数据源控件读取站点信息。SiteMapDataSource 将自动把 Web.sitemap 文件作为站点地图，读取链接数据。

（3）使用导航控件显示导航链接，包括 SiteMapPath 控件、Menu 控件和 TreeView 控件。SiteMapPath 控件常用于在页面头部单行显示当前页的路径；Menu 控件可提供类似下拉列表的链接；TreeView 控件常用于页面的左侧，显示网站中页面的分层。

6.1.2 站点地图

站点地图的创建是实现导航的第一步。站点地图描述了系统中页面的逻辑结构，在

确定了逻辑结构之后，由其他控件直接读取。

网站的逻辑结构表现在页与页之间，每一个大中型网站都会有其页面间的逻辑结构。如一个网上购物系统，首页中包含商品的分类，在首页中选择需要的分类，进入某一个类的商品。

网购中商品信息多，其分类的级别也多，图 6-1 表述了网购系统中的部分页面结构。

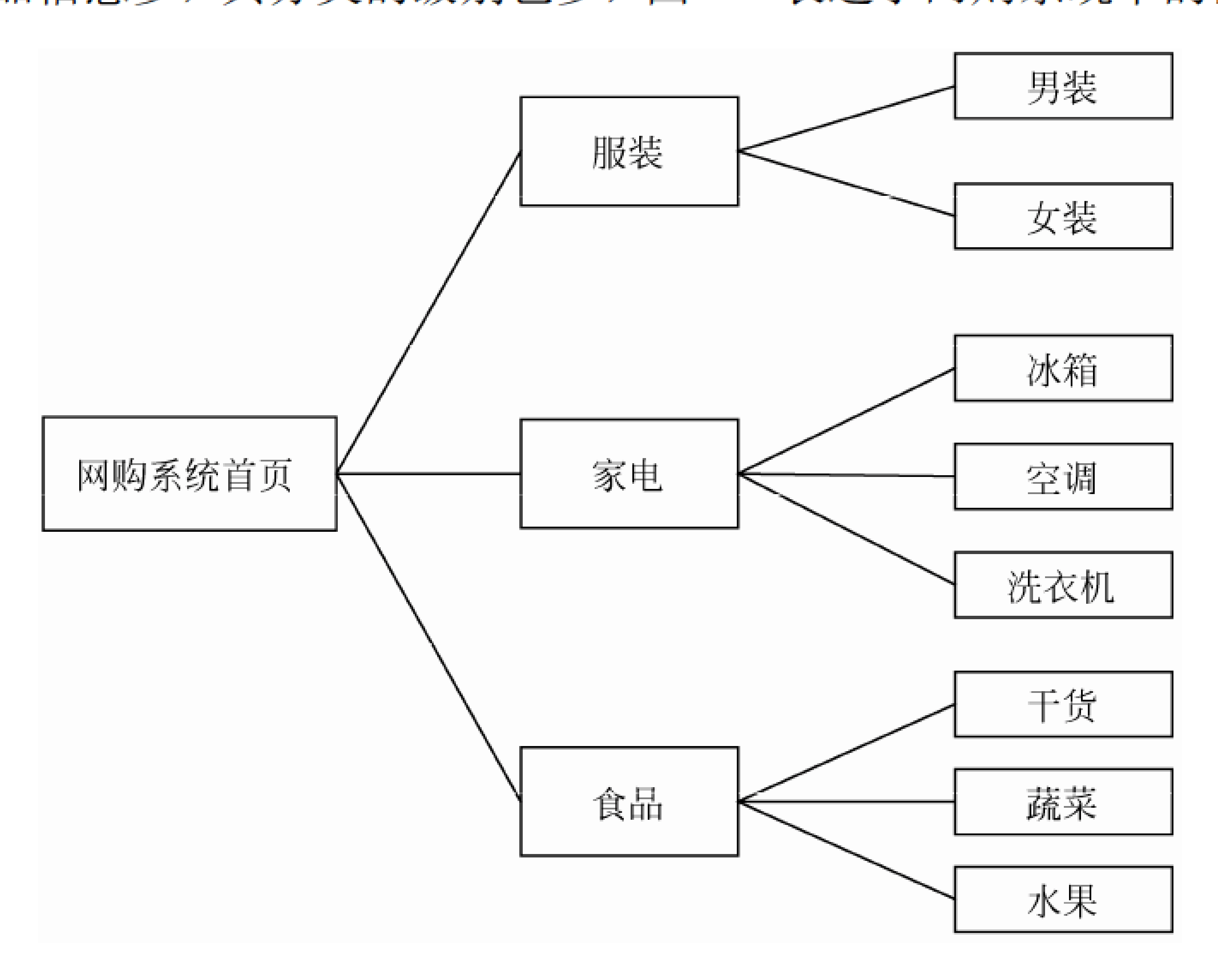


图 6–1　网购系统部分页面逻辑结构

在首页可以选择进入服装类、家电类还是食品类页面，在服装类页面又可以选择进入男装页面还是女装页面。

页面的逻辑结构将页面间的关系和页面执行路径描绘了出来，除了在导航中的作用，对于开发人员来说，对网站的整体结构的认识也变得清晰。

站点地图的创建需要在项目中添加 Web.sitemap 文件，具体方法是，打开【解决方案资源管理器】，在项目名称上右击，选择【添加】|【新建项】命令，打开【添加新项】对话框，如图 6-2 所示。选择【站点地图】选项并单击【添加】按钮，完成站点地图的添加。

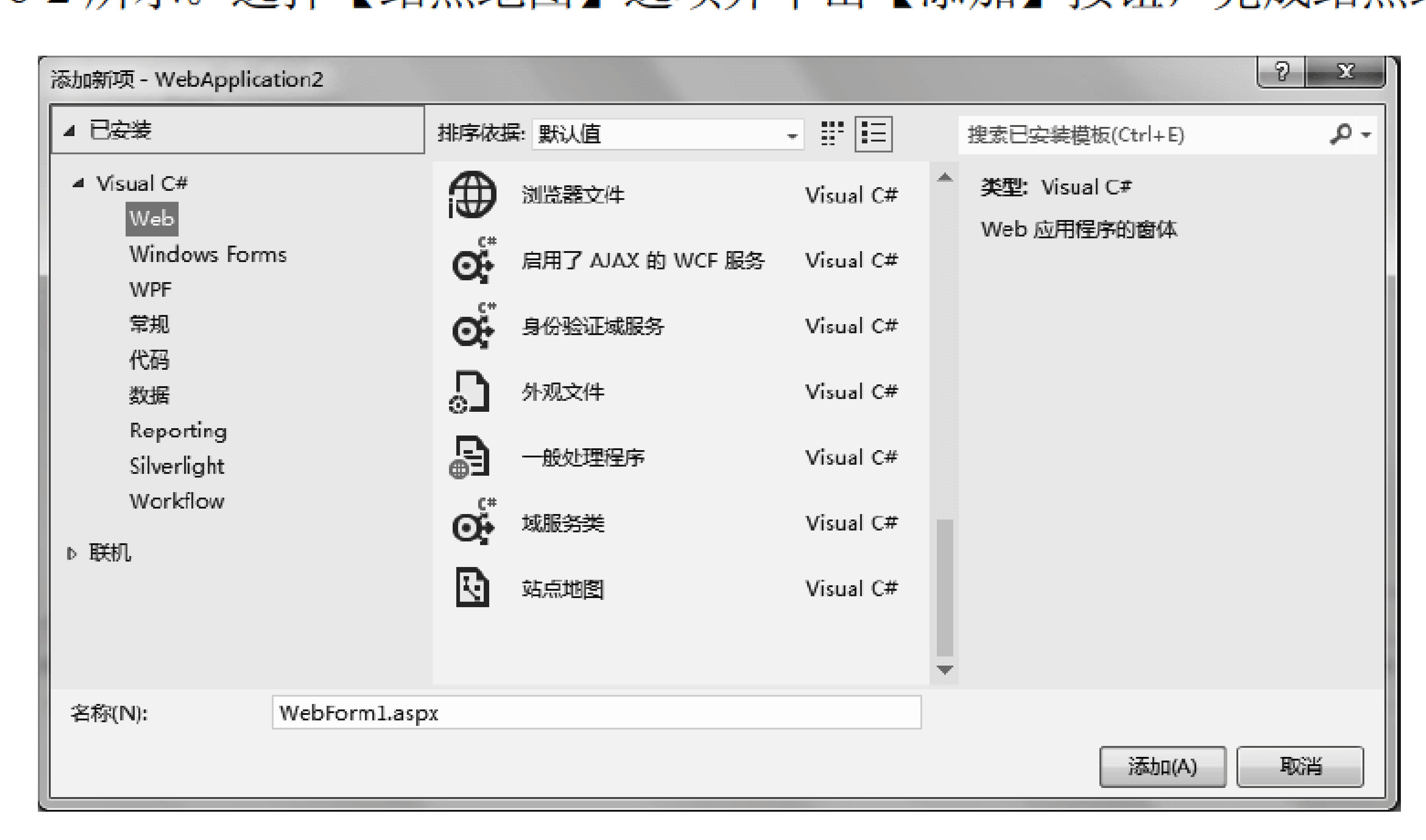


图 6–2　添加站点地图

在站点地图中编辑，可双击站点地图名称，在打开的窗口中进行编辑。新建的站点地图通常有如下代码。

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
   <siteMapNode url="" title=""  description="">
      <siteMapNode url="" title=""  description="" />
      <siteMapNode url="" title=""  description="" />
   </siteMapNode>
</siteMap>
```

对上述代码的解释如下。

（1）sitemap：根节点，一个站点地图只能有一个 siteMap 元素。

（2）siteMapNode：对应于页面的节点，一个节点描述一个页面。

（3）title：页面描述，通常用于指定页面标题。

（4）url：文件在解决方案中的路径。

（5）description：指定链接的描述信息。

虽然 Web.sitemap 文件的内容非常简单，但是编写时需要注意以下几点。

（1）站点地图的根节点为<siteMap>，每个文件有且仅有一个根节点。

（2）<siteMap>下一级有且仅有一个<siteMapNode>节点。

（3）<siteMapNode>下面可以包含多个新的<siteMapNode>节点。

（4）每个站点地图中同一个 URL 只能出现一次。

【范例 1】

如果将如图 6-1 所示的页面逻辑结构在站点地图中描述，则站点地图中代码如下。

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="" title="网购系统首页"  description="首页">
    <siteMapNode url="" title="服装"  description="">
      <siteMapNode url="" title="男装"  description="" />
      <siteMapNode url="" title="女装"  description="" />
    </siteMapNode>
    <siteMapNode url="" title="家电"  description="">
      <siteMapNode url="" title="冰箱"  description="" />
      <siteMapNode url="" title="空调"  description="" />
      <siteMapNode url="" title="洗衣机"  description="" />
    </siteMapNode>
    <siteMapNode url="" title="食品"  description="">
      <siteMapNode url="" title="干货"  description="" />
      <siteMapNode url="" title="蔬菜"  description="" />
      <siteMapNode url="" title="水果"  description="" />
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

每一个有着子节点的<siteMapNode>节点都要以</siteMapNode>结尾，没有子节点的

<siteMapNode>节点可以在其<siteMapNode>尖括号内使用“/”，如<siteMapNode url="Phone.aspx" title="男装"　description="" />。

> **警告**
> Web.sitemap 文件的路径不能更改，必须存放在站点的根目录中，URL 属性必须相对于该根目录。

6.2 SiteMapPath 控件

导航控件通常需要使用数据源控件读取站点地图中的数据，但 SiteMapPath 控件不需要这样，它与站点地图紧密联系，可直接读取站点地图中的数据。Visual Studio 工具下，一个项目有一个默认的站点地图，而 SiteMapPath 控件只能够读取这一个站点地图的信息。

SiteMapPath 控件是使用方式最简单的导航显示控件。SiteMapPath 控件不需要使用 SiteMapDataSource 数据源控件读取站点地图中的数据，在站点地图中包含的页面中直接添加即可显示。

SiteMapPath 控件也叫站点地图导航、痕迹导航或眉毛导航，SiteMapPath 控件显示的是一个导航路径，该路径包含页面的所有上级页面，直至站点地图中的根节点页面。用户可在导航中单击进入上级页面或该导航路径中的其他页面，如图 6-3 所示。



图 6-3 【上衣】的导航路径

图 6-3 描述了网站中由网购首页进入服装页面，之后进入男装页面、上衣页面。通过 SiteMapPath 控件的属性可设置链接的顺序、样式等内容，SiteMapPath 控件常用属性及其说明如表 6-1 所示。

表 6-1 SiteMapPath 控件的常用属性

属性名称	说明
CurrentNodeStyle	获取用于当前节点显示文本的样式
CurrentNodeTemplate	获取或设置一个控件模板，用于代表当前显示页的站点导航路径的节点
NodeStyle	获取用于站点导航路径中所有节点的显示文本样式
NodeStyleTemplate	获取或设置一个控件模板，用于站点导航路径的所有功能站点
ParentLevelsDisplayed	获取或设置控件显示的相对于当前显示节点的父节点级别数
PathDirection	获取或设置导航路径节点的呈现顺序
PathSeparator	获取或设置一个字符串，该字符串在呈现的导航路径中分隔 SiteMapPath 的节点，导航默认的分隔符是“>”
PathSeparatorStyle	获取用于 PathSeparator 字符串的样式
PathSeparatorTemplate	获取或设置一个控件模板，用于站点导航路径的路径分隔符
RootNodeStyle	获取根节点显示文本的样式
RootNodeTemplate	获取或设置一个控件模板，用于站点导航路径的根节点

6.3 TreeView 控件

TreeView 控件是一种应用较为灵活的导航控件，能够以站点地图和 XML 文档作为数据源，还可以在源代码中设置控件的属性和节点显示。

6.3.1 TreeView 简介

TreeView 能够搭建系统的框架，也叫树形视图控件。TreeView 控件以层次或树形结构显示数据，通常与母版页结合，放在网站的左侧作为网站的框架。

XML 格式的文件也可以用作数据源，与站点地图相比，XML 格式文件没有条件限制，只要符合 XML 的标准即可。TreeView 控件常用功能如下。

（1）站点导航，即导航到其他页面的功能。

（2）以文本或链接方式显示节点的内容。

（3）可以将样式或主题应用到控件及其节点。

（4）数据绑定，允许直接将控件的节点绑定到 XML、表格或关系数据源。

（5）为节点实现客户端的功能。

（6）为每一个节点显示复选框按钮。

（7）使用编程方式动态设置控件的属性。

（8）实现节点的添加和填充等。

TreeView 控件若使用站点地图作为数据源，则需要借助 SiteMapDataSource 数据源控件获取站点地图中的数据。由于一个网站中，默认只有一个站点地图，因此只要页面中有 SiteMapDataSource 控件和站点地图，那么 SiteMapDataSource 控件将自动获取站点地图中的数据，不需要开发人员对它们进行绑定。

导航功能需要有页面的逻辑结构和链接，但网站中页面的逻辑结构并不是固定不变的。如网购系统在创建初期只有服装、食品和家电这几个分类，但随着网站做大，需要新增手机数码、家居建材等分类，则需要改变原有的页面逻辑结构。TreeView 控件能够实现节点的添加和填充，需要借助 TreeNode 对象的功能，将在 6.3.3 节中介绍。

TreeView 控件由节点组成，包括父节点、子节点、叶节点和根节点，TreeView 控件的常用属性及其说明如表 6-2 所示。

表 6-2 TreeView 控件的常用属性

属性名称	说明
CheckedNodes	获取 TreeNode 对象的集合，这些对象表示在该控件中显示的选中了复选框的节点
CollapseImageToolTip	获取或设置可折叠节点的指示符所显示的图像的工具提示
CollapseImageUrl	获取或设置自定义图像的 URL，该图像用作可折叠节点的指示符
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
ExpandDepth	获取或设置第一次显示 TreeView 控件时所展开的层次数
ExpandImageToolTip	获取或设置可展开节点的指示符所显示图像的工具提示

续表

属性名称	说明
ExpandImageUrl	获取或设置自定义图像的 URL，该图像用作可展开节点的指示符
LineImagesFolder	获取或设置文件夹的路径，该文件夹包含用于连接子节点和父节点的线条图像
MaxDataBindDepth	获取或设置要绑定到 TreeView 控件的最大树级别数
Nodes	获取 TreeNode 对象的集合，它表示该控件中的根节点
NodeWrap	获取或设置一个值，它指示空间不足时节点中的文本是否换行
NoExpandImageUrl	获取或设置自定义图像的 URL，该图像用作不可展开节点的指示符
PathSeparator	获取或设置用于分隔由 TreeNode.ValuePath 属性指定的节点值的字符
SelectedNode	获取表示该控件中选定节点的 TreeNode 对象
SelectedValue	获取选定节点的值
ShowExpandCollapse	获取或设置一个值，它指示是否显示展开节点指示符

为了动态显示数据，TreeView 控件支持动态节点填充。将 TreeView 控件配置为即需填充时，该控件将在用户展开节点时引发事件。

事件处理程序检索相应数据，然后填充到用户单击的节点。当用户通过一些操作（例如选择、展开或折叠节点）与控件交互时，则会引发 TreeView 控件事件。与属性一样，TreeView 控件包含多个事件，常用事件如表 6-3 所示。

表 6-3　TreeView 控件的常用事件

事件名称	说明
TreeNodeCheckChanged	当 TreeView 控件的复选框发送到服务器的状态更改时发生。每个 TreeNode 对象发生变化时都将发生一次
SelectedNodeChanged	在 TreeView 控件中选定某个节点时发生
TreeNodeExpanded	在 TreeView 控件中展开某个节点时发生
TreeNodeCollapsed	在 TreeView 控件中折叠某个节点时发生
TreeNodePopulate	在 TreeView 控件中展开某个 PopulateOnDemand 属性设置为 true 的节点时发生
TreeNodeDataBound	将数据项绑定到 TreeView 控件中的某个节点时发生

6.3.2　TreeView 简单应用

TreeView 简单应用包括直接应用站点地图中的数据和直接应用 XML 文档中的数据。这里的 XML 文档必须符合站点地图节点特点，以下详细介绍 TreeView 与站点地图的结合和 TreeView 与 XML 文档的结合。

1. TreeView 与站点地图



图 6-4　TreeView 控件

在页面中添加 TreeView 控件，则设计窗口如图 6-4 所示。单击控件右上方的箭头按钮可选择 TreeView 任务，由于页面中没有 SiteMapData

Source 数据源控件，因此 TreeView 任务对话框中没有数据源。

TreeView 借助 SiteMapDataSource 控件和站点地图的使用方式很简单，只需要在页面中添加 SiteMapDataSource，并为 TreeView 绑定该 SiteMapDataSource 即可。如对范例 1 中的站点地图进行修改，在男装节点下新增【上衣】和【裤子】子节点，修改部分代码如下所示。

```
<!--省略部分代码，参考范例 1-->
<siteMapNode url="" title="男装"  description="">
   <siteMapNode url="cloth.aspx" title="上衣"  description="" />
   <siteMapNode url="" title="裤子"  description="" />
</siteMapNode>
<!--省略部分代码，参考范例 1-->
```

【范例 2】

在项目中新建 cloth.aspx 页面，省略页面代码。向页面中添加 SiteMapPath 控件和 TreeView 控件，显示上述站点地图中的页面逻辑，查看它们的显示效果，步骤如下。

（1）向页面中添加 SiteMapPath 控件，只需将控件从工具箱中拉到设计界面即可。其页面代码如下所示。

```
<asp:SiteMapPath ID="SiteMapPath1" runat="server"></asp:SiteMapPath>
```


（2）TreeView 控件需要借助 SiteMapDataSource 控件来获取数据，因此需要先添加 SiteMapDataSource 控件，该控件自动绑定站点地图，代码如下。

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

（3）接着需要添加 TreeView 控件，并绑定上述 SiteMapDataSource。绑定时只需单击 TreeView 控件右上端的箭头按钮，在选择数据源下拉框中选中上述 SiteMapDataSource1 即可，如图 6-5 所示。

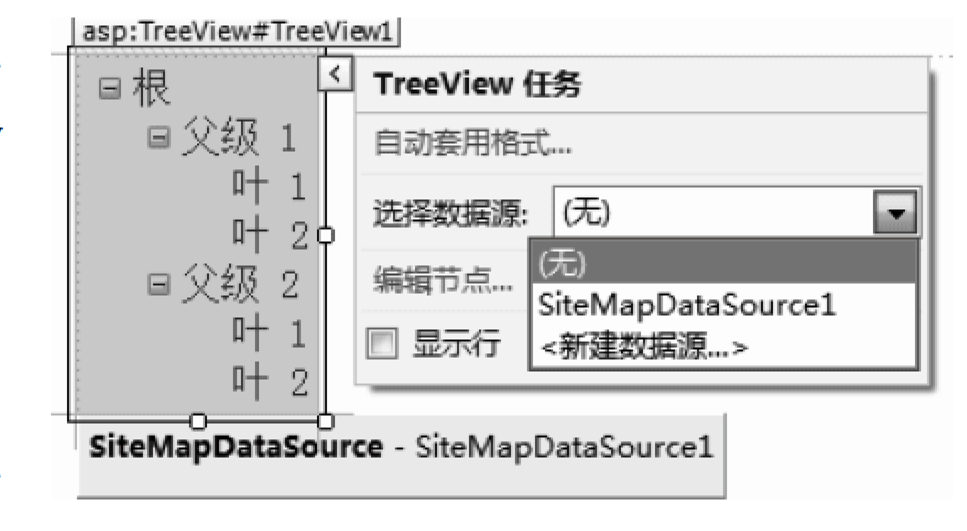


图 6-5 为 TreeView 选择数据源

（4）选择好数据源的 TreeView 控件，其页面代码如下。

```
<asp:TreeView  ID="TreeView1"  runat="server"  DataSourceID="SiteMapData
Source1" Width="194px"></asp:TreeView>
```

（5）运行该页面，其效果如图 6-6 所示。在页面的左侧中间位置，有 SiteMapPath 控件所显示的样式：网购系统首页>服装>男装>上衣，可单击链接进入上层页面。在页面的左侧下方位置有 TreeView 控件样式，呈现站点地图中的所有页面。

2. TreeView 与 XML 文档

除了可以显示站点地图数据，TreeView 控件还可以显示 XML 文档数据。这里以范例的形式介绍 XML 文档的使用。



图 6-6 SiteMapPath 控件和 TreeView 控件的显示效果

【范例 3】

定义 XML 文档内容为网站的页面逻辑，修改图 6-6 中的 TreeView 控件使其显示 XML 文档中的页面结构，步骤如下。

（1）首先向项目中添加 XML 文档，在项目名称处右击，在【添加新项】窗体左侧选择【数据】选项，右侧选择【XML 文件】选项，为文件命名并单击【添加】按钮完成添加。为了区分 XML 文档与站点地图，定义与站点地图不一样的页面结构，其 XML 文档代码如下。

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMapNode url="" title="网购系统首页" description="首页">
   <siteMapNode url="" title="服装" description="">
      <siteMapNode url="" title="男装" description="">
         <siteMapNode url="cloth.aspx" title="上衣" description="" />
         <siteMapNode url="" title="裤子" description="" />
      </siteMapNode>
      <siteMapNode url="" title="女装" description="">
         <siteMapNode url="" title="上衣" description="" />
         <siteMapNode url="" title="裤子" description="" />
         <siteMapNode url="" title="连体裤" description="" />
         <siteMapNode url="" title="连衣裙" description="" />
      </siteMapNode>
      <siteMapNode url="" title="内衣" description="" />
   </siteMapNode>
   <siteMapNode url="" title="家电" description="">
      <!--省略部分代码-->
```

```
        </siteMapNode>
    </siteMapNode>
```

（2）修改 TreeView 控件的数据源，在如图 6-5 所示的【选择数据源】下拉框中选择【新建数据源】，打开【数据源配置向导】对话框，如图 6-7 所示。选择 XML 文档将自动创建一个数据源 ID，单击【确定】按钮后进入【配置数据源】对话框，如图 6-7 所示。单击【数据文件】文本框右侧的【浏览】按钮，即可进入选择 XML 文档的页面，选择 XML 文档后确定，即可绑定 TreeView 控件与 XML 文档。



图 6–7　选择 XML 文档

（3）在绑定了 TreeView 控件与 XML 文档之后，系统并不会显示 XML 文档中的页面结构，这是因为 XML 文档的页面节点并没有固定的格式要求。如节点名称可以是任意名称，而不需要像站点地图一样使用 siteMap 或 siteMapNode；同时页面的标题和链接属性也不需要像站点地图一样使用 title 和 url。因此在绑定了文档之后，需要对节点的标题和链接属性进行绑定，否则系统因无法识别页面的属性而仅显示节点名称，如图 6-8 所示。

（4）如图 6-8 所示，选择【TreeView 任务】面板中的【编辑 TreeNode 数据绑定】项，可进入【TreeView DataBindings 编辑器】对话框，如图 6-8 所示。在【可用数据绑定】区域内选择需要绑定的节点并单击【添加】按钮，可在右侧看到该节点的属性。其中，TextField 属性对应页面的标题，选择 XML 文档中对应的 title 属性，如图 6-8 所示；NavigateUrlField 属性对应页面的链接地址属性，选择 XML 文档中对应的 url。在单击【确定】按钮后可回到页面的设计窗体看到被绑定的页面效果。



图 6-8 XML 文档绑定

（5）运行该页面，其效果如图 6-9 所示。页面的链接默认使用蓝色字体显示。



图 6-9 TreeView 绑定 XML 文档

6.3.3 TreeNode 对象

TreeNode 对象本身可以理解为 TreeView 的节点，因此 TreeView 控件对节点的操作可以理解为对 TreeNode 对象的操作。

TreeNode 对象在执行中有两种模式：选择模式和导航模式。这是 TreeView 控件节点特有的，内容如下。

（1）选择模式：单击节点会回发页面并引发 TreeView.SelectedNodeChanged 事件，这是默认的模式。

（2）导航模式：单击后导航到新页面，不会触发上述事件。只要 NavigateUrl 属性非空，TreeNode 就会处于导航模式。

由于站点地图中，每一个<siteMapNode>节点都要提供一个 URL 信息，因此绑定到站点地图的 TreeNode 都属于导航模式。页面的逻辑结构要求页面节点也有着逻辑结构，节点中也有着与节点结构相关的名称，如“父节点”和“子节点”，如下所示。

（1）包含其他节点的节点称为“父节点”。

（2）被其他节点包含的节点称为“子节点”。

（3）没有子节点的节点称为“叶节点”。

（4）不被其他任何节点包含同时是所有其他节点的上级的节点是“根节点”。

一个节点可以同时是父节点和子节点，但是不能同时为根节点、父节点和叶节点。节点为根节点、子节点还是叶节点决定着节点的几种可视化属性和行为属性。

虽然一个树结构中只有一个根节点，但是 TreeView 控件允许开发人员向树结构中添加多个根节点。当用户要显示项目列表，但不显示单个主根节点时（如产品类别列表），这一功能很有用。

页面节点有着独特的属性，如页面的标题、链接、节点图片等属性，如表 6-4 所示。

表 6-4 TreeNode 对象的常用属性

属性名称	说明
Text	获取或设置控件中节点的文本
Value	获取或设置控件中节点的值
Checked	获取或设置一个值，该值指示节点的复选框是否被选中
ChildNodes	获取 TreeNodeCollections 集合，该集合表示第一级节点的子节点
DataItem	获取绑定到控件的数据项
Depth	获取节点的深度
Expanded	获取或设置一个值，该值指示是否展开节点
ImageUrl	获取或设置节点旁显示的图像的 URL
NavigateUrl	获取或设置单击节点时导航到的 URL
ShowCheckBox	获取或设置一个值，该值指示是否在节点旁显示一个复选框
SelectAction	获取或设置选择节点时引发的事件
Selected	获取或设置一个值，该值指示是否选择节点
Target	获取或设置用来显示与节点关联的网页内容的目标窗口或框架

单击 TreeView 控件的节点将引发选择事件（通过回发）或导航至其他页，如果未设置 NavigateUrl 属性，单击某个节点将引发 SelectedNodeChanged 事件，该事件可用于提供自定义功能。

【范例 4】

向 TreeView 控件中添加根节点，页面的标题是“会员中心”，页面的地址是“vip.aspx”，

代码如下。

```
TreeNode tn = new TreeNode();           //创建节点
tn.Text ="会员中心";                     //为节点名称赋值
tn.NavigateUrl = "vip.aspx";
TreeView1.Nodes.Add(tn);                //将节点添加为 TreeView1 的根节点
```

上述代码将新增一个与“网购系统首页”相同等级的页面，相当于两个根元素。这样的添加方式只是在页面中显示新节点，并没有更新到数据源文件中。

6.3.4 TreeView 样式

TreeView 有一个细化的样式模型，通过对样式的设置，可以完全控制 TreeView 的外观。

TreeView 的样式控制可以具体到一个节点，其样式的控制由 TreeNodeStyle 类来实现。TreeNodeStyle 继承自更常规的 Style 类。除了设置导航的前景色、背景色、字体和边框，还能引入如表 6-5 所示的样式属性。

表 6-5 TreeNodeStyle 样式属性

属性名称	说明
ImageUrl	节点旁边显示的图片
NodeSpacing	当前节点与相邻节点的垂直距离
VerticalPadding	节点文字与节点边界内部的垂直距离
HorizontalPadding	节点文字与节点边界内部的水平距离
ChildNodesPadding	展开的父节点的最后一个子节点和其下一个兄弟节点的间距

TreeView 导航节点可以用表格呈现，因此可以设置文字边距和节点间距。除了在表 6-5 中列举的属性以外，TreeView 还有以下几个高级属性。

（1）TreeView.NodeIndent 属性：用于设置树结构里各个子层级间缩进的像素数。

（2）TreeView.ShowExpandCollapse 属性：用于关闭树中的节点列。

（3）TreeView.ShowCheckBoxes 属性：设置节点边是否显示复选框。

（4）TreeNode.ShowCheckBox 属性：设置单个节点边是否显示复选框。

要对树的所有节点应用样式，可以使用 TreeView.NodeStyle 属性，而要为特定的节点设置样式，需要首先选择节点。TreeView 控件支持节点层次的样式，如父节点层次的样式与子节点层次上的样式不同。节点层次的选择需要使用如表 6-6 所示的属性。

表 6-6 节点层次的选择

属性名称	说明
NodeStyle	应用到所有节点
RootNodeStyle	仅应用到第一层（根）节点
ParentNodeStyle	应用到所有包含其他节点的节点，但不包括根节点
LeafNodeStyle	应用到所有不包含子节点而且不是根节点的节点
SelectedNodeStyle	应用到当前选中的节点
HoverNodeStyle	应用到鼠标停留的节点.

样式的执行优先级与页面元素的样式优先级类似，遵循从通用到特定的规律，如 SelectedNodeStyle 样式将覆盖 RootNodeStyle 的样式设置。

另外，通过 TreeViewNode.ImageUrl 可以为节点设置图片。通常需要为整个树形导航设置一组风格一致的图片，因此可定义的节点属性有 4 种，如表 6-7 所示。

表 6-7 节点指示符属性

属性名称	说明
CollapseImageUrl	可折叠节点的指示符所显示的图像。默认为一个减号
ExpandImageUrl	可展开节点的指示符所显示的图像。默认为一个加号
LineImagesFolder	包含用于连接父节点和子节点的线条图像的文件夹的图像
NoExpandImageUrl	不可展开节点的指示符所显示的图像

ASP.NET 系统中提供了 16 种内置的节点图片以供选择，可在设计界面中，TreeView 控件的 ImageSet 属性中直接选择。

为 TreeView 控件分配图像的最简单方法是使用 ImageSet 属性。内置于 TreeView 控件中的图像集包括树中用于 MSN Messenger、Microsoft Outlook、Windows Explorer 和 Microsoft Windows 帮助的常见图像资源集。图像集还包括几种项目符号样式，可以通过获取 ImageSet 属性查看所有的图像集。

6.4 Menu 控件

Menu 控件又称作动态菜单控件，是一种可动态显示的控件。Menu 控件有两种显示模式：动态模式和静态模式。

（1）静态模式表示 Menu 控件始终是完全展开的，及所有的导航节点都是可视的，用户可直接单击进入。

（2）动态模式表示只有指定的部分是静态可视的，被合并的节点通常在用户移动鼠标至父节点时显示。

这两种模式并不是独立的，而是可以相互结合的。Menu 控件的数据来源可以绑定数据源，也可以在控件中直接添加。Menu 控件的属性及其说明如表 6-8 所示。

表 6-8 Menu 控件的常用属性

属性名称	说明
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DynamicBottomSeparatorImageUrl	获取或设置图像的 URL，该图像显示在各动态菜单项底部，将动态菜单项与其他菜单项隔开
DynamicEnableDefaultPopOutImage	获取或设置一个值，该值指示是否显示内置图像，其中内置图像指示动态菜单项具有子菜单
DynamicHorizontalOffset	获取或设置动态菜单相对于其父菜单项的水平移动像素数
Items	获取 MenuItemCollection 对象，该对象包含 Menu 控件中的所有菜单项
ItemWrap	获取或设置一个值，该值指示菜单项的文本是否换行
MaximumDynamicDisplayLevels	获取或设置动态菜单的菜单呈现级别数

续表

属性名称	说明
Orientation	获取或设置 Menu 控件的呈现方向，默认值是 Vertical
PathSeparator	获取或设置用于分隔 Menu 控件的菜单项路径的字符
ScrollDownText	获取或设置 ScrollDownImageUrl 属性中指定的图像的替换文字
ScrollUpText	获取或设置 ScrollUpImageUrl 属性中指定的图像的替换文字
SelectedItem	获取选中的菜单项
SelectedValue	获取选中菜单项的值
StaticEnableDefaultPopOutOutImage	获取或设置一个值，该值指示是否显示为内置图像，其中内置图像指示静态菜单项包含的子菜单项
DisappearAfter	获取或设置鼠标指针不再置于菜单上后显示动态菜单的持续

Menu 控件与 TreeView 控件一样支持层次化数据，既可以绑定数据源，又可以使用 MenuItem 对象填充数据。Menu 对数据源的绑定可参考 TreeView 控件。

与 TreeNode 对象相比，MenuItem 对象的样式较为简单，没有复选框，也不能通过编程来控制节点的展开和折叠。MenuItem 的属性如表 6-9 所示。

表 6-9 MenuItem 的属性

属性名称	说明
Text	菜单中显示的文字
ToolTip	鼠标停留菜单项时的提示文字
Value	保存不显示的额外数据（比如某些程序需要用到的 ID）
NavigateUrl	如果设置了值，单击节点会前进至此 Url。否则，需要响应 Menu.MenuItemClick 事件确定要执行的活动
Target	它设置了链接的目标窗口或框架。Menu 自身也暴露了 Target 属性设置所有的 MenuItem 实例的默认目标
Selectable	如果为 false，菜单项不可选。通常只在菜单项有一些可选的子菜单项时，才设为 false
ImageUrl	菜单项旁边的图片
PopOutImageUrl	菜单项包含子项时显示在菜单项旁的图片，默认是一个小的实心箭头
SeparatorImageUrl	菜单项下面显示的图片，用于分隔菜单项

Menu 控件和 TreeView 控件的呈现方式非常不同，但它们有着相似的编程模型。两者的具体区别如下。

（1）Menu 每次显示一个子菜单；TreeView 可以一次展开任意多个节点。

（2）Menu 在页面里显示第一层的链接；TreeView 显示页面上内联的所有项。

（3）Menu 不支持按需填充及客户端回调；TreeView 支持。

（4）Menu 支持模板；TreeView 不支持。

（5）Menu 支持水平和垂直布局；TreeView 只支持垂直布局。

（6）Menu 控件的样式属性同样比较完善。与 TreeView 控件一样，Menu 从 Style 基类派生了自定义类，除了与 TreeView 控件类似的属性外，还增加了节点的间距属性。

（7）Menu 控件没有 ImageUrl 属性。

Menu 控件样式在很大程度上和 TreeView 样式相似，它支持不同层级的节点使用不

同的菜单样式。但是，由于 Menu 控件的显示方式有着动态和静态之分，因此 Menu 控件定义了两组功能类似的属性，如表 6-10 所示。

表 6-10　Menu 控件显示样式

静态样式	动态样式	说明
StaticMenuStyle	DynamicMenuStyle	设置总体“盒子”的外观，所有的菜单项出现在这里
StaticMenuItemStyle	DynamicMenuItemStyle	设置单个菜单项的外观
StaticSelectedStyle	DynamicSelectedStyle	设置选择项的外观，选择项指的是前一个被单击的项
StaticHoverStyle	DynamicHoverStyle	设置鼠标停留时项的外观

Menu 控件对特定层次上的样式通过集合的方式进行设置，包括 LevelMenuItemStyles 属性、LevelSubMenuStyles 属性和 LevelSelectedStyles 属性。这些集合分别作用于普通的菜单项，包含其他菜单项的菜单项以及被选择的菜单项。

Menu 控件一个有趣的样式是 StaticDisplayLevels，它允许设置静态层次的数目。静态层次默认只有一层，需要在 Menu.StaticDisplayLevels 属性中设置。当该属性值大于 1 时，还可通过 StaticSubMenuIdent 属性控制每层的缩进。

注意

如果将 MaximumDynamicDisplayLevels 的属性值设置为 0 则不会动态显示任何菜单节点；如果将该属性的值设置为负数则会引发异常。

设置的层次越多，所能设置的相关属性也越多，包括菜单消失前的延时（DisappearAfter）、展开图标和分隔符的默认图片和滚动行为等。

Menu 控件还可以支配模板以控制每个菜单要呈现的样式，通过 StaticMenuItemTemplate 和 DynamicMenuItemTemplate 属性，无论是以声明的方式还是编程的方式填充 Menu 类，都能够使用模板。



从模板的角度来看，必须要绑定到 MenuItem 对象，再获取节点的值。

Menu 控件在页面的顶部以样式块的形式呈现其所有样式，而不与呈现的 HTML 内联。但是，可以把 Menu.IncludeStyleBlock 属性设为 false 以告知 Menu 不要呈现其样式，这样能够完全控制 Menu 样式，甚至可以采用外部样式表的样式。

注意

Menu 控件不提倡使用 HTML 表格来呈现，而是把节点呈现为一组无序的条目（使用<ul>和<li>元素），并通过样式规则来创建正确的格式。

6.5 实验指导——男裤选购页面

以图 6-10 为例，创建男裤页面 trousers.aspx，在站点地图中为该页面节点添加链接，

站点地图代码省略，要求如下。

（1）页面中有 SiteMapPath 控件、Menu 控件和 TreeView 控件。

（2）Menu 控件放在页面的头部右上角，显示为横着展开的动态模式，初始状态下只显示一级节点。

（3）TreeView 控件所有节点字体颜色使用黑色。

（4）TreeView 控件根节点使用开心购物图片 gouwu.JPG 作为节点图片。

（5）TreeView 控件父节点使用“购”字图片 gou.JPG 作为节点图片。

实现上述要求，步骤如下。

（1）SiteMapPath 控件与上衣页面中的一样，代码省略。使用 Menu 控件显示为横着展开的动态模式，初始状态下只显示一级节点，代码如下。

```
<asp:Menu ID="Menu1" runat="server" Orientation="Vertical" StaticDisplay
Levels="1" DataSourceID="SiteMapDataSource1"></asp:Menu>
```

（2）TreeView 控件所有节点字体颜色使用黑色；根节点使用开心购物图片 gouwu.JPG 作为节点图片；父节点使用“购”字图片 gou.JPG 作为节点图片，代码如下。

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
<asp:TreeView  ID="TreeView1"  runat="server"  DataSourceID="SiteMap
DataSource1" Width="194px">
    <RootNodeStyle  ImageUrl="~/Images/gouwu.JPG" NodeSpacing="7px"/>
    <NodeStyle ForeColor="Black"/>
    <ParentNodeStyle ImageUrl="~/Images/gou.JPG" />
</asp:TreeView>
```

（3）运行上述页面，其效果如图 6-10 所示。鼠标放在 Menu 控件上，可看到展开的部分节点。



图 6-10 男裤页面

思考与练习

一、填空题

1．能够自动绑定站点地图的导航控件是__________。

2．站点地图的文件名是__________。

3．站点地图只能有一个__________元素。

4．常用的导航数据源有__________和站点地图。

二、选择题

1．下列控件中，有着动态和静态模式的是__________。

A．TreeView 控件

B．Menu 控件

C．SiteMapPath 控件

D．SiteMapDataSource 控件

2．关于导航控件下面说法错误的是__________。

A．SiteMapPath 控件实现的导航功能很简单，只能够将站点地图作为数据源

B．TreeView 控件又叫树形菜单导航控件，它可以由一个或多个节点组成，节点可以使用 TreeNode 对象表示

C．Menu 控件有静态模式和动态模式两种，这两种模式可以一起使用

D．SiteMapPath、Menu 和 TreeView 控件都可以将 XML 文件和站点地图文件作为数据源

3．与 TreeNode 对象用法相似的是_______。

A．MenuItem 对象

B．TreeView 对象

C．SiteMapPath 对象

D．SiteMapDataSource 对象

4．下列不属于站点地图节点的默认属性的是__________。

A．url

B．title

C．text

D．description

三、简答题

1．总结 TreeView 控件的应用。

2．简单说明导航控件与数据源文件的绑定方式。

3．总结 TreeView 控件与 Menu 控件的区别。

第 7 章　使用母版页

一个网站往往有多个页面，每个页面都要有样式和页面布局。通常一个网站系统中的页面使用同一种样式和布局，在页面的特定位置放置页面的特有内容；这种样式和布局就成为网站的风格。这种风格被称为页面的主题。

使用母版页是实现网站中统一页面样式和布局的手段，母版页相当于为页面设计了一个模板，所有页面只需要在母版页的基础上编写各自的特有内容即可。

本章主要介绍母版页的使用，主题的应用，以及创建和使用用户控件。

本章学习要点：

- ❑ 理解母版页和内容页的关系
- ❑ 理解母版页的使用原理
- ❑ 掌握外观文件的使用
- ❑ 掌握主题的加载
- ❑ 能够创建和使用用户控件
- ❑ 理解用户控件的转换

7.1 母版页

母版页的作用相当于网站的模板，在母版页基础上编写的页面被称作内容页，母版页和内容页共同构成了有着统一风格的网站系统。本节介绍母版页和内容页的相关概念和应用。

7.1.1 母版页概述

母版页能够将页面上的公共元素（如系统网站的 Logo、导航条和广告条等）整合到一起用来创建一个通用的外观，它的优点如下。

（1）有利于站点修改和维护，降低开发人员的工作强度。

（2）提供高校的内容整合能力。

（3）有利于实现页面布局。

（4）提供一种便于利用的对象模型。

母版页以“.master”作为后缀名，是 master 文件。母版页的页面与一般的 Web 窗体非常相似，可以添加 HTML 元素和服务器端控件，其内容的编辑与一般的 Web 页面一样。

风格一样的页面，也有着相同的部分和不同的部分。母版页是将网站中页面的相同部分定义下来，并指定一个区域供内容页编写其特有内容。

在母版页中，为内容页预留的区域使用 ContentPlaceHolder 控件占据着，一个母版页中可以预留一个或多个 ContentPlaceHolder。

由于母版页与一般的 Web 页面相似，因此可通过它们之间的区别，来分析母版页的特点，如下所示。

（1）母版页使用@Master 指令，而 Web 窗体页使用@Page 指令。

（2）母版页可以使用一个或多个 ContentPlaceHolder 控件，用来占据一定的空间，而 Web 窗体页则不允许使用 ContentPlaceHolder 控件。

（3）母版页派生自 MasterPage 类，而 Web 窗体页派生自 Page 类。

（4）母版页后缀名是.master，普通页面后缀名为.aspx。

（5）母版页仅作为网站设计时的中介文件，而 Web 窗体页是可直接在浏览器中显示的文件。

在介绍母版页的创建之前，首先了解一下当前网站的常见布局方式。母版页是作为网页的一部分被呈现，而母版区域的划分有着多种布局形式，常见布局如下。

（1）栏式结构布局。栏式结构是很常见的一种结构，它简单实用、条理分明并且格局清晰严谨，适合信息量大的页面。常见的几种栏式布局如图 7-1 所示。

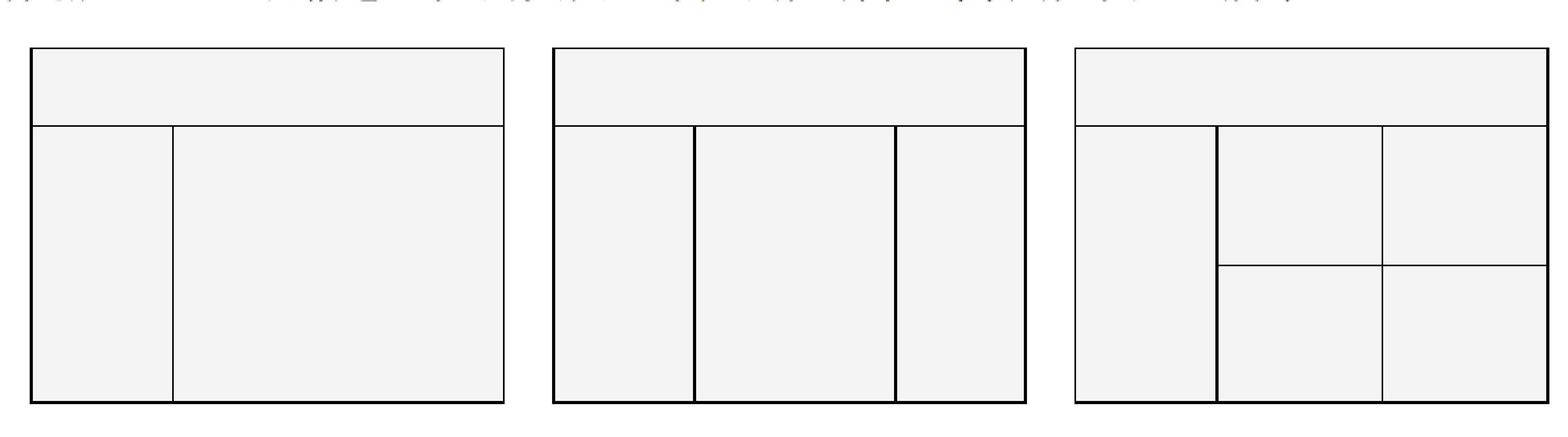


图 7-1 常见的栏式布局

（2）区域结构布局。区域结构的特点是页面精美、主题突出以及空间感很强。但是它适合信息量比较少的页面，并且在国内使用的比较少。区域机构可以将页面分隔成若干个区域，如图 7-2 所示。

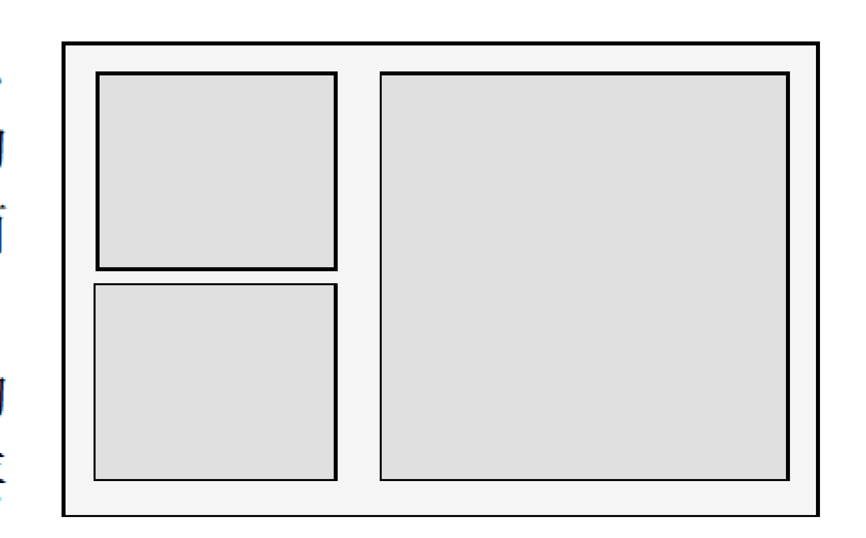


图 7-2 区域结构示例

如图 7-1 和图 7-2 所示，母版页区域常用在页面的上部、左侧或同时用于上部和左侧。母版页的创建需要在项目下添加新建项，与站点地图的添加步骤一样，如图 7-3 所示。

7.1.2 添加内容页

内容页与一般的 Web 页面一样，是 aspx 文件。其添加方式与一般的 Web 页面一样，只是在添加窗口中需要选择【使用母版页的 Web 窗体】选项，打开【选择母版页】对话框，如图 7-4 所示。选择需要的母版页，单击【确定】按钮完成创建。



图 7-3 添加母版页



图 7-4 选择母版页

内容页的创建有多种方式，上述创建是其中一种。如在母版页中直接创建内容页，可在母版页中右击，选择【添加内容页】命令即可添加该母版页的内容页。添加完成之后，内容页中代码如下所示。

```
<%@ Page Title="" Language="C#" MasterPageFile="~/SiteHead.Master"
AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebAppli
cation1.WebForm1" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
runat="server">
</asp:Content>
```

代码的首行 MasterPageFile="~/SiteHead.Master"语句定义了该内容页所归属的母版页，后面的两个 Content 控件对应母版页中的两个 ContentPlaceHolder 控件，控件在页面中所占据的位置可参考对应的母版页。

除了直接创建内容页，还可将现有的非内容页改为内容页，步骤如下。

（1）页面代码中添加 MasterPageFile="母版页地址"语句，指定所归属的母版页。

（2）删除页面中 Content 控件区域以外的标记，即页面中的所有标记都放在 Content 控件内部，并删除<html>、<head>和<body>标记。这些页面标记是 Web 页面所需要的，但内容页需要嵌套在母版页中，因此不需要这些标记。

7.2 实验指导——内容页与母版页的结合

本节通过实验指导的方式，介绍内容页与母版页的结合。首先定义一个母版页，包括头部部分和左侧部分，接着创建该母版页的两个内容页，查看内容页的执行效果，步骤如下。

（1）首先设计母版页，将页面分成顶部、左侧和右侧三部分，省略各个部分的控件代码，页面的布局代码如下。


```
<body>
    <form id="form1" runat="server">
        <div style="margin-left: 60px; width: 820px">
            <div id="headDiv">
                    <%--代码省略--%>
            </div>
            <div >
                <div id="leftDiv" style="border: thin solid #ED145B; float:
                left; width: 190px;">
                    <%--代码省略--%>
                </div>
                <div id="rightDiv" style="float: left;">
                    <div >图片展示</div>
                    <div style="margin-left: 20px">
                        <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
                       runat="server">
                        </asp:ContentPlaceHolder>
                    </div>
                </div>
            </div>
        </div>
    </form>
</body>
```

（2）上述代码的设计效果如图 7-5 所示。



图 7-5　母版页设计

（3）接下来添加该母版页的内容页，并在页面中添加高山图片，代码如下。

```
<asp:Content  ID="Content2"  ContentPlaceHolderID="ContentPlaceHolder1"
runat="server">
    <asp:Image ID="Image1" runat="server" Height="380" Width="600"
    ImageUrl="~/Images/hill.jpg" />
</asp:Content>
```

（4）运行该内容页，其效果如图 7-6 所示。虽然页面中仅添加了一个图片，但同时加载了母版页的页面综合了母版页与内容页。



图 7-6　高山内容页

（5）添加流水内容页，同样只在页面中添加一个图片控件，代码如下。

```
<asp:Content  ID="Content2"  ContentPlaceHolderID="ContentPlaceHolder1"
runat="server">
    <asp:Image ID="Image1" runat="server" Height="380" Width="600" ImageUrl
    ="~/Images/lake.jpg" />
</asp:Content>
```

（6）运行该页面，其效果如图 7-7 所示。与图 7-6 相比，只有 Content 控件的位置发生了变化。



图 7-7　流水内容页

7.3　主题

主题是另一种统一页面风格的方式，母版页设置页面所共有的控件和布局；而主题设置页面的外观样式，包括背景颜色、字体颜色和边框等。主题和母版页共同控制着页面的风格，本节介绍主题的概念及其应用。

7.3.1　主题与外观文件

主题常用来定义一个系统的统一样式或风格，它与母版页都可以作用在多个页面上。主题有着以下几个特点。

（1）主题可以由多个文件组成，包括样式表、外观、图片或文件等。

（2）页面的主题可以在运行中替换掉。

（3）主题可以应用于多个页面，也可以用于多个应用程序。

无论主题是页主题还是全局主题，主题中的内容都是相同的。主题中包含三个重要的概念：主题、外观和样式表。其具体说明如下。

（1）主题（Theme）：它是一组属性，包括外观文件、级联样式表（CSS）文件、图像等元素，它可以将这些元素应用于服务器控件并规定其样式。

（2）外观（Skin）：外观文件的后缀名是.skin，它包含各个服务器控件的属性设置。外观文件也叫作皮肤文件。

（3）样式表（CSS）：样式表文件定义控件和块的样式，包括颜色、位置、对齐方式等。

主题并不是一个独立的文件，而是由多个定义样式和外观的文件构成的文件夹。主题的添加需要在项目名称上右击，如图 7-8 所示。

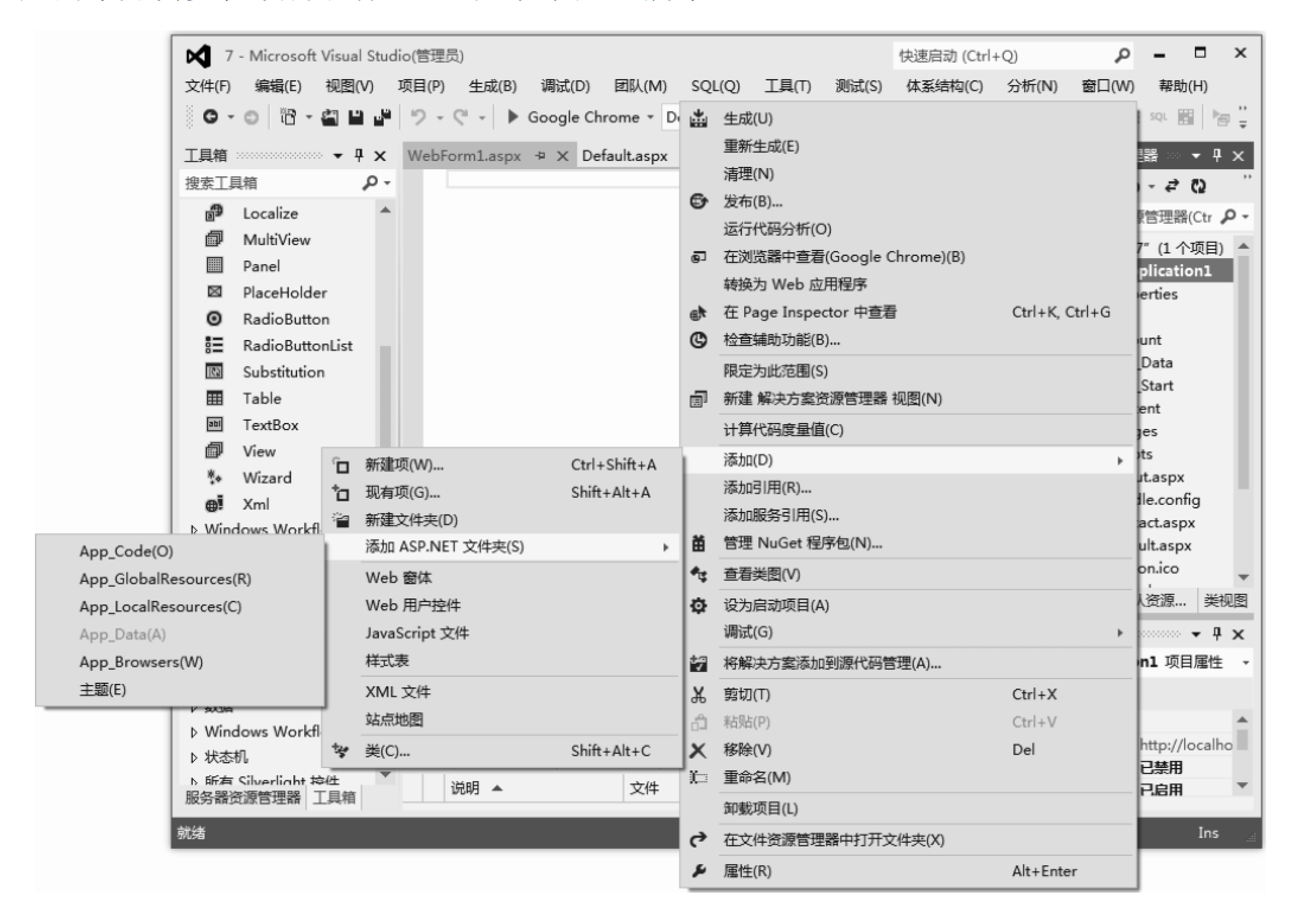


图 7-8 添加主题

分别选择【添加】|【添加 ASP.NET 文件夹】|【主题】选项添加主题。主题将默认被添加在 App_Themes 文件夹下。接着在该主题文件夹下添加外观文件和样式文件等，完成主题创建。

外观文件的作用与样式表类似，但外观文件只定义页面中控件的样式。控件获取样式表中的样式，使用 class 等属性；而获取主题的样式，使用 SkinID 属性。

外观文件即为主题中的皮肤，其文件中的样式定义有两种形式，如下所示。

（1）不含 SkinID 属性，则其定义的样式将应用于所有控件。

（2）含 SkinID 属性，只有指定了该 SkinID 的控件才遵循指定的样式。

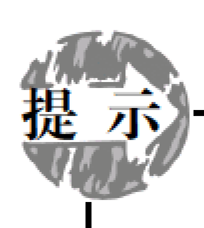
不含 SkinID 属性的控件样式又称为默认样式。主题中定义了 SkinID 属性的控件，可在页面中通过 SkinID 属性为页面控件指定样式。

如定义一个 TextBox 控件样式不含 SkinID 属性，则引用了该样式的页面中，所有的 TextBox 控件均为该样式。

主题文件的编写与控件的代码类似，不同的是，主题文件中的控件不能有 ID 属性；而且若没有特殊要求，最好没有 Text 属性。否则该控件不能起作用或只能显示同样的 Text 值。

如分别定义一个默认的 TextBox 控件样式，用来使 TextBox 字体颜色为黑色；以及一个指定了 SkinID 属性的控件样式，使其字体颜色为白色，其代码如下所示。

```
<%--默认样式--%>
<asp:TextBox runat="server" Font-Size="Large" ForeColor="Black"></asp:
TextBox>
<%--指定样式--%>
<asp:TextBox SkinId="White" runat="server" Font-Size="Large" ForeColor="White">
</asp:TextBox>
```

样式表中的注释与页面注释语法一样，使用<%-- --%> 标签来编辑注释内容。

7.3.2 主题的创建

一个网站系统可以有多个主题，主题可以在页面创建后创建，也可在页面创建前创建，因此主题创建后并不是默认被页面加载的。

在 ASP.NET 中可以通过多种方式加载主题，如在页面中设置 Theme 或 StylesheetTheme 属性、通过配置文件以及通过改变页面的 Theme 属性值、SkinID 属性值或 CssClass 属性值动态加载等。下面详细介绍如何使用这些方式加载主题。

1. 通过修改配置文件为多个页面批量加载主题

一个网站系统只有一个 Web.config 配置文件，因此使用配置文件进行主题加载，该主题将被用于所有页面。

在 Web.config 中添加 Theme 属性或者 StylesheetTheme 属性，需要放在<system.web>节点下，即在文件中使用如下代码。

```
<configuration>
  <system.web>
    <pages styleSheetTheme="主题名称或目录"/>
  </system.web>
</configuration>
```

注意

> 在配置文件目录下设置页面主题时必须去掉页面中@Page 指令里的 Theme 属性或者 StylesheetTheme，否则会重写配置文件中的对应属性。

2. 通过改变页面的 Theme 属性值动态加载主题

使用配置文件加载主题，能够使主题在页面加载之前被加载。而若要在页面中动态加载或改变主题，则需要使用页面的 PreInit 事件。这时主题中的皮肤文件和样式表文件会同时被加载。

Page 对象中的事件由系统自动加载，其中，PreInit 事件在 Load 事件之前发生，而页面及其控件的主题必须在 PreInit 事件或 PreInit 事件之前发生。Page 对象中部分事件的发生顺序如下所示。

（1）PreInit 事件，在页初始化开始时发生。

（2）Init 事件，当服务器控件初始化时发生；初始化是控件生存期的第一步。

（3）InitComplete 事件，在页初始化完成时发生。

（4）PreLoad 事件，在页 Load 事件之前发生。

（5）Load 事件，当服务器控件加载到 Page 对象中时发生。

（6）LoadComplete 事件，在页生命周期的加载阶段结束时发生。

（7）PreRender 事件，在加载 Control 对象之后、呈现之前发生。

（8）PreRenderComplete 事件，在呈现页内容之前发生。

在 PreInit 事件中对主题进行加载时，除了可以加载页面的主题，还可加载单个控件的主题。

如对 TextBox 控件加载主题文件中主题 SkinID="White"的样式，使用语句如下所示。

```
protected void Page PreInit(object sender,EventArgse)
{
    TextBox.SkinID ="White";
}
```



> 在 PreInit 事件中加载主题中的皮肤，其皮肤文件必须是含 SkinID 属性的皮肤。

3. 通过改变控件的 CssClass 属性值动态加载主题中的样式表

除了动态加载主题和动态加载主题中的皮肤外，还可以在后台页面中直接通过控件的 CssClass 属性值动态加载主题中的样式表。

注意

> 母版页中不能定义主题，所以不能在@Master 指令中使用 Theme 属性或 StylesheetTheme 属性。如果需要集中定义所有页面的主题，可以通过在 Web.config 文件中配置来实现。

在页面的 Page 指令中添加 Theme 或 StylesheetTheme 属性都可以用来加载指定的主题。但是当主题中不包含皮肤文件时两者的效果都一样，当主题中包含皮肤文件时两者因为优先级不一样会产生不一样的效果。它们的优先级依次为：StylesheetTheme>Page>Theme。

当加载主题到页面后，因为某些原因需要禁用某个页面或某个控件的主题，这时候可以通过设置 Theme 或 StylesheetTheme 的值为空来完成。另外，还可以将控件的 EnableTheming 的属性值设置为 false 指定禁用主题中的皮肤。



读者可以通过添加新的案例比较 Theme 和 StylesheetTheme 的不同。

7.4 实验指导——主题切换

本节通过实验指导的方式，介绍主题的创建、加载和切换。借用 7.2 节实验指导中的母版页样式，创建一个没有色彩的 ASP.NET 网页，添加【红色主题】和【紫色主题】两个按钮。分别创建两个主题，定义两种色彩样式，并在页面中通过按钮切换网页的主题颜色，步骤如下。

（1）首先参照 7.2 节实验指导中的母版页定义 ASP.NET 网页，代码省略。其设计效果如图 7-9 所示。



图 7–9　页面设计效果

（2）如图 7-9 所示，与图 7-7 相比，页面中几乎都是黑白样式，没有色彩。接下来定义与图 7-7 色彩一致的红色主题，主题名为 red，在主题中创建样式表和外观文件，代码如下。

样式表代码：

```
div {
    color: #ED145B;
}
```

外观文件代码：

```
<asp:Button runat="server" BackColor="#ED145B" ForeColor="White" />
<asp:TreeView runat="server" Width="180px" BackColor="White">
    <NodeStyle ForeColor="Black" BackColor="White" />
    <LeafNodeStyle ImageUrl="~/Images/pic.jpg" ForeColor="#ED145B" />
 </asp:TreeView>
<asp:Panel runat="server" BackColor="#ED145B"> </asp:Panel>
```

（3）之后定义紫色主题，与红色主题相比，修改了元素的颜色，其样式表代码和外观文件代码如下。

样式表代码：

```
div {
    color: BlueViolet
}
```

外观文件代码：

```
<asp:Button runat="server" BackColor="BlueViolet" ForeColor="White" />
<asp:TreeView runat="server" Width="180px" BackColor="White">
    <NodeStyle ForeColor="Black" BackColor="White" />
    <LeafNodeStyle ImageUrl="~/Images/pic.jpg" BackColor="White" ForeColor=
    "BlueViolet" />
 </asp:TreeView>
<asp:Panel runat="server" BackColor="BlueViolet"> </asp:Panel>
```

（4）最后定义页面的后台代码，在页面加载时默认加载红色主题，在单击【红色主题】和【紫色主题】两个按钮后分别替换为红色主题和紫色主题。

由于主题的加载是需要在页面的 PreInit 事件中执行，那么【红色主题】和【紫色主题】两个按钮执行主题的切换需要重新加载页面，同时向页面中传入参数，在 PreInit 事件中根据传入的参数加载相应的主题。

页面中需要定义的事件有两个按钮的单击事件和 PreInit 事件，如下所示。

① 红色主题按钮的代码：

```
protected void Red Click(object sender, EventArgs e)
{
    Response.Redirect("WebForm1.aspx?colour=red");
}
```

② 紫色主题按钮的代码：

```
protected void BlueViolet Click(object sender, EventArgs e)
{
```

```
    Response.Redirect("WebForm1.aspx?colour=BlueViolet");
}
```

③ PreInit 事件的代码：

```
public partial class WebForm1 : System.Web.UI.Page
{
    protected void Page_PreInit(object sender, EventArgs e)
    {
        if (!string.IsNullOrEmpty(Request.QueryString["colour"]))
                                                    //判断是否有主题数据
        {
            if (Request.QueryString["colour"] == "red")
                                                    //判断主题数据是否为红色
            { Theme = "red"; }
            else
            { Theme = "BlueViolet"; }
        }
        else
        {
            Theme = "red";
        }
    }
}
```


（5）运行该页面，其效果如图 7-10 所示。该页面与图 7-7 显示的效果相似。单击【紫色主题】按钮，其效果如图 7-11 所示。若单击【红色主题】按钮，可回到如图 7-10 所示的页面。



图 7-10 红色主题



图 7-11 紫色主题

7.5 用户控件

有编程经验的用户应该知道，程序开发语言有着代码的可重用性，通常使用类和函数来实现。页面中的控件组合也有着可重用性，除 Visual Studio 工具箱中的控件以外，可以使用用于创建 ASP.NET 网页的相同技术创建可重复使用的自定义控件。这些控件称作用户控件。

7.5.1 用户控件概述

用户控件是一种复合控件，工作原理非常类似于 ASP.NET 网页。可以向用户控件添加现有的 Web 服务器控件和标记，并定义控件的属性和方法；然后可以将控件嵌入 ASP.NET 网页中充当一个控件。

有时可能需要控件中具有内置 Web 服务器控件未提供的功能。在这种情况下有两种控件可以自定义创建，如下所示。

（1）用户控件。用户控件是能够在其中放置标记和 Web 服务器控件的容器。用户控件创建之后可以作为一个控件对待，为其定义属性和方法。

（2）自定义控件。自定义控件是编写的一个类，此类从 Control 或 WebControl 派生。

创建用户控件要比创建自定义控件方便很多，因为可以重用现有的控件。用户控件

使创建具有复杂用户界面元素的控件极为方便。

用户控件与完整的 ASP.NET 网页（aspx 文件）相似，同时具有用户界面页和代码。可以采取与创建 ASP.NET 页相似的方式创建用户控件，然后向其中添加所需的标记和子控件。用户控件可以像页面一样包含对其内容进行操作（包括执行数据绑定等任务）的代码。用户控件与 ASP.NET 网页有以下区别。

（1）用户控件的文件扩展名为.ascx。

（2）用户控件中没有@Page 指令，而是有着@Control 指令，该指令对配置和属性进行定义。

（3）用户控件不能作为独立文件运行。而必须像处理其他控件一样，将它们添加到 ASP.NET 页中。

（4）用户控件中没有 html、body 或 form 元素，这些元素必须位于宿主页中。

可以在用户控件上使用与在 ASP.NET 网页上所用相同的 HTML 元素（html、body 或 form 元素除外）和 Web 控件。用户控件与 ASP.NET 页十分相像，它包含若干控件、处理按钮的 Click 事件和页面的 Load 事件的代码。

7.5.2 创建用户控件

创建 ASP.NET 用户控件的方法与设计 ASP.NET 网页的方法极为相似。在标准 ASP.NET 页上使用的 HTML 元素和控件也可用在用户控件上。但是，用户控件没有 html、body 和 form 元素，并且文件扩展名必须为.ascx。

在项目中创建用户控件与创建母版页的方法一样，在项目名称处右击，选择添加新建项可打开【添加新项】对话框，选择【Web 用户控件】即可。向新用户控件添加任何标记和控件，并为该用户控件将执行的所有任务（例如，处理控件事件或从数据源读取数据）添加代码。

用户控件的属性和方法的定义，可参考页面的属性和方法的定义。通过定义用户控件的属性，就能以声明方式或代码方式设置其属性。

用户控件包含 Web 服务器控件时，可以在用户控件中编写代码来处理其子控件引发的事件。例如，如果用户控件包含一个 Button 控件，则可以在用户控件中为该按钮的 Click 事件创建处理程序。

默认情况下，用户控件中的子控件引发的事件对于宿主页不可用。但是，可以为用户控件定义事件并引发这些事件，以便将子控件引发的事件通知宿主页。

用户控件支持独立于宿主页的缓存指令。因此，可以向页面添加用户控件，并对页面的某些部分进行缓存。

用户控件创建后可以在 ASP.NET 网页中添加，但是在添加之前需要在 ASP.NET 网页中进行注册。注册用户控件时要指定包含用户控件的.ascx 文件、标记前缀以及将用于在页面上声明用户控件的标记名称。

注册用户控件需要在 ASP.NET 网页中创建一个@Register 指令，包括如下三个属性。

（1）TagPrefix 属性：该属性定义用户控件的前缀与用户控件相关联，此前缀将用在

用户控件的开始标记中。

（2）TagName 属性：该属性定义用户控件的名称，此名称将用在用户控件元素的开始标记中。

（3）Src 属性：该属性定义用户控件文件的虚拟路径。Src 属性值既可以是相对路径，也可以是从应用程序的根目录到用户控件源文件的绝对路径。为灵活使用，建议使用相对路径。代字号（~）表示应用程序的根目录。用户控件不能位于 App_Code 目录中。

在网页主体中，可以在 form 元素内部声明用户控件元素。如果用户控件公开公共属性，要以声明方式设置这些属性。

【范例 1】

大多网页中都有着关键字的搜索，如新闻页面中可根据关键字搜索相关新闻，购物网页中可以根据关键字搜索商品。创建一个用户控件用于关键字的搜索，要求包含一个文本框和一个按钮，步骤如下。

（1）首先创建用户控件文件名为 WebSelect.ascx，步骤代码省略。向文件中添加服务器控件，为了页面效果，将文本框和按钮放在表中。用户控件代码如下。

```
    <%@        Control        Language="C#"        AutoEventWireup="true"
CodeBehind="WebSelect.ascx.cs" Inherits="WebApplication1.WebSelect" %>
    <table>
        <tr style="background-color: #CCFFFF; width: 280px; height: 30px;
text-align: center">
            <td style="border: medium solid #CCFFFF; width: 70%;">
                <asp:TextBox ID="TextKey" runat="server" BorderStyle="None"
Height="22px" Width="188px"></asp:TextBox></td>
            <td style="width: 30%">
                <asp:Button ID="ButSel" runat="server" Text="搜索" Width="92px"
BackColor="#CCFFFF" BorderStyle="None" Font-Size="Large" /></td>
        </tr>
    </table>
```

（2）创建 ASP.NET 网页，注册上述用户控件并将用户控件放在页面中，注册指令代码如下。

```
    <%@ Register src="~/WebSelect.ascx" tagname="WebSelect" tagprefix="sel" %>
```

（3）向页面中添加上述用户控件，代码如下。

```
    <sel:WebSelect ID="Select1" runat="server" />
```

上述代码中，控件的前缀与步骤（2）中的 tagprefix 属性值对应；控件名 WebSelect 与步骤（2）中的 tagname 属性名对应。

7.5.3 ASP.NET 用户控件转换

如果已经开发了 ASP.NET 网页并打算在整个应用程序中访问其功能，则可以对该页面略加改动，将它更改为一个用户控件。将 ASP.NET 网页转换为用户控件有如下几个

步骤。

（1）重命名控件使其文件扩展名为.ascx。

（2）从该页面中移除 html、body 和 form 元素。

（3）将@Page 指令更改为@Control 指令。

（4）移除@Control 指令中除 Language、AutoEventWireup、CodeFile 和 Inherits 之外的所有属性。

（5）在@Control 指令中添加 className 属性。这允许将用户控件添加到页面时对其进行强类型化。

除此之外，还可以将有着代码隐藏文件的 ASP.NET 网页转换为用户控件，步骤如下。

（1）重命名.aspx 文件，使其文件扩展名为.ascx。

（2）根据代码隐藏文件使用的编程语言，重命名代码隐藏文件使其文件扩展名为.ascx.vb 或.ascx.cs。

（3）打开代码隐藏文件并将该文件继承的类从 Page 更改为 UserControl。

（4）.aspx 文件中移除 html、body 和 form 元素。

（5）将.aspx 文件中的@Page 指令更改为@Control 指令。

（6）移除.aspx 文件中的@Control 指令中除 Language、AutoEventWireup、CodeFile 和 Inherits 之外的所有特性。

（7）在.aspx 文件中的@Control 指令中，将 CodeFile 属性更改为指向重命名的代码隐藏文件。

（8）在.aspx 文件中的@Control 指令中添加 className 属性。这允许将用户控件添加到页面时对其进行强类型化。

思考与练习

一、填空题

1．母版页的后缀名是__________。

2．母版页使用__________指令，而 Web 窗体页使用@ Page 指令。

3．母版页可以使用一个或多个__________控件代表内容页。

4．内容页的后缀名是__________。

5．皮肤文件的后缀名是__________。

6．Web.config 中添加 Theme 属性或者 StylesheetTheme 属性，需要放在__________节点下。

二、选择题

1．切换主题的本质是改变页面的__________属性值。

A．PreInit

B．EnableTheming

C．Page

D．Theme

2．皮肤文件又称作是__________文件。

A．样式表

B．外观

C．主题

D．母版页

3．主题的动态加载，需要在页面的__________事件中执行。

A．PreInit

B．Load

C．InitComplete

D．PreLoad

4．下列说法错误的是__________。

A．通过将控件的 EnableTheming 属性的值设置为 false 可以禁用主题中的

皮肤

B．StylesheetTheme 的优先级高于 Theme

C．StylesheetTheme 的优先级低于 Theme

D．主题一般有两种形式：页主题和全局主题

5．下列不属于@Register 指令必需的属性是__________。

A．TagPrefix

B．TagName

C．TagSrc

D．Src

三、简答题

1．简述母版页和内容页是如何结合的。

2．简单说明主题的构成。

3．总结主题的加载方式。

4．简述用户控件的使用。

第 8 章　验证用户输入的有效性

在 Web 应用程序中，通常会接收大量的用户输入数据。在这些数据中，可能会出现一些无效的数据，甚至是恶意的数据，这将给 Web 应用程序的安全带来威胁。因此，需要对用户输入的数据进行验证。数据验证是验证用户数据真实性和正确性的过程，用以鉴别用户输入的数据是否合法。

在 ASP.NET 中编写 Web 应用程序时，用户保存或者处理信息就需要判断其有效性和安全性。本章着重介绍 ASP.NET 中的验证控件，通过验证控件对用户输入的数据进行有效性和安全性验证。

本章学习要点：

- ❑ 了解 ASP.NET 中常用的数据验证技术
- ❑ 掌握必填验证控件的属性和使用
- ❑ 掌握比较验证控件的属性和使用
- ❑ 掌握范围验证控件的属性和使用
- ❑ 熟悉正则表达式验证控件的属性和使用
- ❑ 掌握错误验证汇总控件的使用
- ❑ 掌握 ValidationGroup 属性的使用

8.1 常用的数据验证技术

由于 Web 应用程序是基于请求/响应模式的，所以 Web 的数据验证有多种方式。可以在服务器端直接对数据进行验证，也可以编写客户端脚本来实现数据有效性的验证，当这些数据提交给服务器时就经过了验证。在实际的项目开发中，既需要客户端验证，也需要服务器验证。

8.1.1 基于图片和附加码的验证

目前，开发者实现图片验证码时有两种方式：一种是通过动态数据网页中的各种脚本来实现；一种是用支持动态数据网页的第三方组件来实现。组件的应用会提高效率，比较容易实现。在 ASP.NET 中编写基本的脚本，赋予其必要的属性，例如生成码颜色、码位数和码尺寸等，就可以灵活地生成一组验证码。

验证码图片一般放在用户名和用户密码的后边，可以根据需要放置。附加码通常由服务器随机产生，一般是由数字和字母组成的一串字符，显示在登录页面中，用户登录时必须将附加码一并输入提交，服务器对提交的验证码同时进行验证。

8.1.2 Web 表单数据验证

在 ASP.NET 中，被指定为 runat="server"的表单被称为 Web 表单，Web 表单本身是基于服务器的，是 ASP.NET 用来为应用程序提供大部分功能框架的一部分，服务器对界面的情况一清二楚，也就是说用户元素只能在服务器上创建。当用户输入完数据提交表单时，服务器将通过另外的页面来验证 Web 表单中所输入数据是否有效。

表单在实现验证方面具有灵活性和易于实现性，其数据验证功能比较强大，开发者既可以把用户信息放在 Web.config 配置文件中，也可以将用户的验证信息放在数据库或 XML 文件中，通过创建自己定义的程序来验证数据。

8.1.3 Web 窗体页数据验证

在 ASP.NET 中提供了一种新型的数据验证，它将使用 Web 窗体页中的 Web 服务器控件来实现数据验证，通常将其称为 Web 窗体页数据验证技术。因此，把专门用于 Web 数据验证的 Web 服务器控件称为 Web 数据验证控件。

ASP.NET 中提供了 6 种数据验证控件，5 种基础验证控件，1 种验证汇总控件，这些控件将在 8.2 节和 8.3 节进行介绍。

8.1.4 客户端脚本验证

为了减少数据验证时浏览器和服务器之间的往返时间，可以采用客户端脚本来实现其功能，在浏览器中使用的脚本有很多，如 VBScript、JScript 和 JavaScript 等，但是这样存在安全隐患。这是因为用户可以任意修改客户端脚本跳过客户端的验证，还有些浏览器是不支持客户端脚本的，这样，数据验证就必须在服务器进行。

8.1.5 使用正则表达式进行数据验证

正则表达式（Regular Expression）是由普通字符（称为原义字符）和特殊字符（称为元字符）组成的字符串，用来定义字符处理时需要匹配的内容模式。也就是说，正则表达式可以让用户通过使用一系列的特殊字符构建匹配模式，然后把匹配模式与 Web 页面的表单输入等目标对象进行比较，根据比较对象中是否包含匹配模式以执行相应的处理操作。对于处理字符串的许多应用程序而言，正则表达式是不可缺少的模式描述工具。

8.2 基础验证控件

ASP.NET 中提供了 6 种服务器端的数据验证控件，其中包括 5 种基础验证控件和 1 种汇总控件。下面分别介绍这 5 种基础验证控件，包括必填验证控件、比较验证控件、

范围验证控件、正则表达式验证控件以及自定义验证控件。

8.2.1 必填验证控件

在 ASP.NET 中，RequiredFieldValidator 控件表示必填验证控件。必填验证控件用于确保用户不会跳过某一项输入。因此，可以将必填验证控件称为非空验证控件。例如，用户在登录页面进行登录时，必须输入用户名和密码，这时可以使用 RequiredFieldValidator 控件验证它们必须填写。

注 意

无论是 RequiredFieldValidator 控件还是下面介绍的其他控件，它们都需要与另一个控件配合使用，不能单独使用。首先将要验证的控件添加到网页中，然后再添加验证控件，这样就可以轻松地将后者与前者关联。

简单来说，RequiredFieldValidator 控件就是检查是否有输入值，语法如下。

```
<ASP:RequiredFieldValidator id="Validator_Name" Runat="Server" ControlTo
Validate="要检查的控件名" ErrorMessage="出错信息" Display="Static|
Dymatic|None">
    占位符
</ASP: RequiredFieldValidator>
```

在上述语法中，“占位符”表示 Display 属性的值为 Static 时，错误信息占有“占位符”那么大的页面空间。另外，从上述代码可以看出，RequiredFieldValidator 控件与其他控件一样，也可以为其指定属性。除了上述属性外，它还可以包含其他属性，如表 8-1 所示。

表 8-1 RequiredFieldValidator 控件的常用属性

属性名称	说明
ControlToValidate	获取或设置要验证的控件 ID 名称。此属性必须进行设置
ErrorMessage	获取或设置验证失败时 ValidationSummary 控件中显示的错误消息的文本
Text	获取或设置验证失败时验证控件中显示的文本
Display	控件错误消息的显示方式。它的值包括三个，说明如下。 （1）Static：表示控件的错误信息在页面中占有肯定位置 （2）Dynamic：表示控件错误信息出现时才占用页面控件 （3）None：表示错误出现时不显示，但是可以在 ValidatorSummary 中显示
ValidationGroup	获取或设置此验证控件所属的验证组的名称
IsValid	获取或设置一个值，该值指示输入的控件是否通过验证
InitialValue	获取或设置关联的输入控件的初始值
EnableClientScript	获取或设置一个值，该值指示是否启用客户端验证。默认为 false
SetFocusOnError	获取或设置一个值，该值指示验证失败时是否将焦点设置到第一个验证失败的控件上

【范例 1】

设计登录页面，该页面包含【用户名】输入框和【密码】输入框，通过 Required

FieldValidator 控件指定输入框的内容必须填写。页面代码如下。

```
<table width="100%" height="180" align="center">
    <tr>
        <td align="right" class="auto-style1">用户名: </td>
        <td><asp:TextBox ID="txtUserName" runat="server" CssClass="frame1">
        </asp:TextBox></td>
        <td><asp:RequiredFieldValidator ID="rfvUserName" runat="server"
        ControlToValidate="txtUserName">用户名不能为空</asp:RequiredField
        Validator> </td>
    </tr>
    <tr>
        <td align="right" class="auto-style1">密码: </td>
        <td><asp:TextBox ID="txtUserPass" runat="server" TextMode=
        "Password" CssClass="frame1"></asp:TextBox></td>
        <td><asp:RequiredFieldValidator ID="rfvUserPass" runat="server"
        ControlToValidate="txtUserPass">密码不能为空</asp:RequiredField
        Validator> </td>
    </tr>
    <tr>
        <td class="auto-style1"></td>
        <td><asp:Button ID="btnLogin" runat="server" CssClass="login1"
        /></td>
    </tr>
</table>
```

在上述代码中，通过 RequiredFieldValidator 控件分别验证【用户名】输入框和【密码】输入框的内容必须填写。

运行上述代码查看效果，如图 8-1 所示。



图 8-1 登录页面

除了通过“占位符”的方式设置提示内容，还可以通过 Text 属性和 ErrorMessage 属性进行设置。如果“占位符”和 Text 属性同时设置，那么先显示“占位符”指定的内容。ErrorMessage 属性的值表示验证失败时 ValidationSummary 控件中显示的错误文本。如果 ErrorMessage 属性和 Text 属性同时存在，则单击按钮时会显示 Text 属性的值；如果没有设置 Text 属性，而设置 ErrorMessage 属性，则单击按钮时会显示 ErrorMessage 属性的值。

【范例 2】

更改范例 1 中 RequiredFieldValidator 控件的内容，为第一个必填验证控件指定 Text 属性，为第二个必填验证控件指定 ErrorMessage 属性和 Text 属性。相关代码如下。

```
<asp:RequiredFieldValidator  ID="rfvUserName"  runat="server"  Control
ToValidate="txtUserName"  ErrorMessage=" 必  填 "></asp:RequiredField
Validator>
<asp:RequiredFieldValidator ID="rfvUserPass" runat="server" ControlTo
Validate="txtUserPass" ErrorMessage=" 必 填 " Text=" 必 须 填 写 "></asp:
RequiredFieldValidator>
```

重新运行页面，单击页面中的【登录】按钮进行测试，如图 8-2 所示。

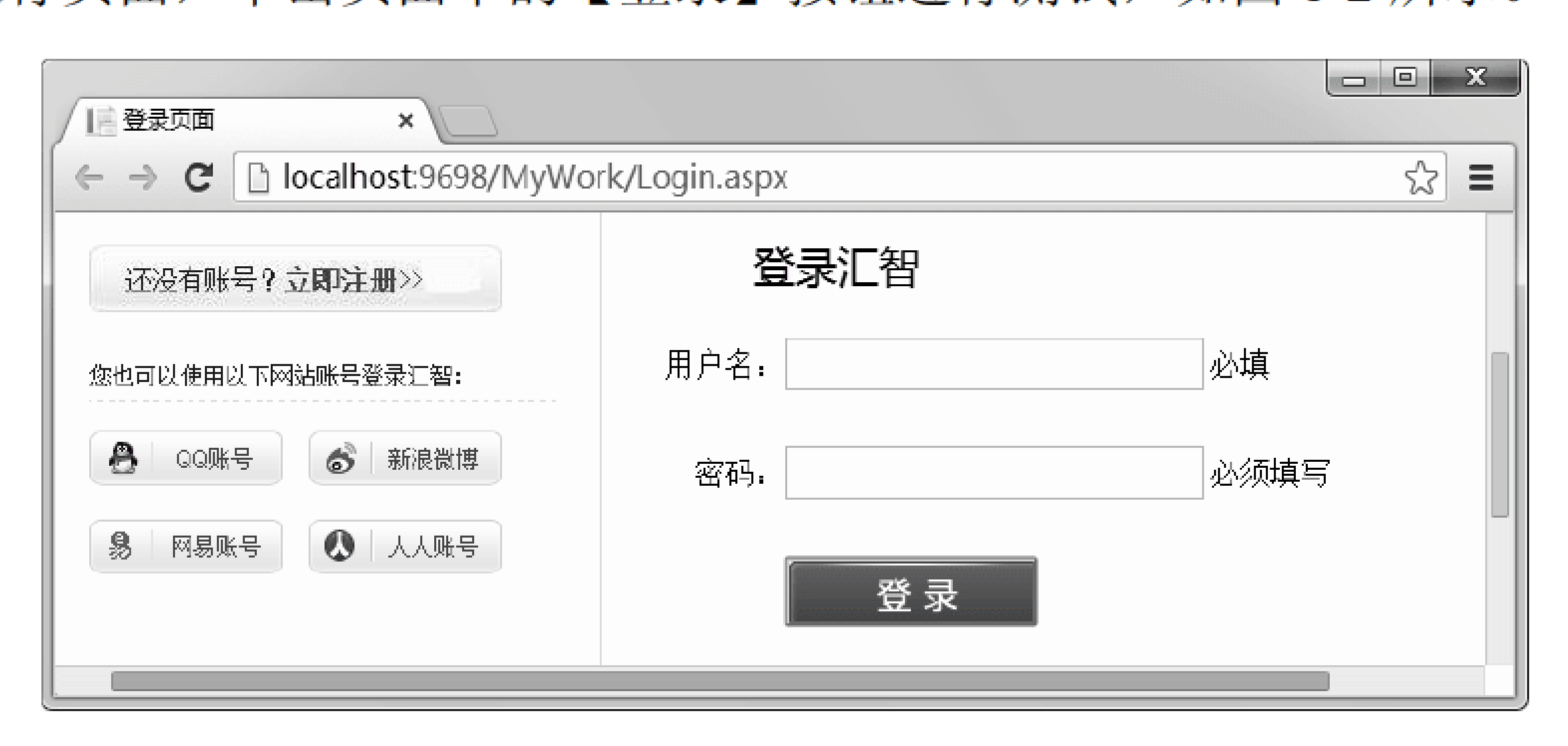


图 8-2 指定 Text 属性和 ErrorMessage 属性

8.2.2 比较验证控件

在 ASP.NET 中，CompareValidator 表示比较验证控件。比较验证控件可以将用户输入的一个常数值与另一个控件或者特定数据类型的值进行比较，比较时需要使用小于、等于或者大于等比较运算符。

CompareValidator 控件包含多个属性，它的大多数属性都与 RequiredFieldValidator 控件一样，如 ControlToValidate 属性、Text 属性、ErrorMessage 属性和 Display 属性等。除了这些属性外，CompareValidator 控件还包含一些特有属性，如表 8-2 所示。

表 8-2 CompareValidator 控件的特有属性

属性名称	说明
ControlToCompare	获取或设置要与所验证的输入控件进行比较的输入控件
Operator	获取或设置要执行的比较操作，默认值为 Equal
Type	获取或设置在比较之前所比较的值转换到的数据类型。其值包括 String（默认值）、Integer、Double、Date 和 Currency
ValueToCompare	获取或设置一个常数值，该值要与由用户输入到所验证的输入控件中的值进行比较

在表 8-2 中，Operator 属性的值是枚举类型 ValidationCompareOperator 的值之一，该

枚举类型的值有 7 个，它们分别是 Equal（默认值，等值比较）、DataTypeCheck（只对数据类型进行比较）、GreaterThan（大于比较）、GreaterThanEqual（大于或等于比较）、LessThan（小于比较）、LessThanEqual（小于或等于比较）和 NotEqual（不等于比较）。

> **注意**
> 如果在 ControlToCompare 属性中指定控件，CompareValidator 控件会将用户输入的内容与指定的控件进行比较。如果 ControlToCompare 和 ValueToCompare 属性同时指定了值，则 ControlToCompare 的优先级比较高。

1．控件值与控件值比较

用户在网站进行注册时，需要输入注册成功后的登录密码，为了安全，通常会要求用户重新确认密码。可以通过 CompareValidator 控件比较用户输入的值，将确认密码输入框的值与密码输入框的值进行比较，下面通过范例说明。

【范例 3】

设计用户注册页面，该页面包含【用户名】输入框、【密码】和【确认密码】输入框以及【您的邮箱】输入框。通过 RequiredFieldValidator 控件指定所有输入框的内容必填，CompareValidator 控件比较【密码】输入框和【确认密码】输入框的值。主要代码如下。

```
<table width="100%" height="300">
    <tr>
        <td align="right" width="20%">用户名：</td>
        <td><asp:TextBox ID="txtUserName" runat="server" CssClass="frame-
        gray1"></asp:TextBox><asp:RequiredFieldValidator ID="rfvUserName"
        runat="server" ControlToValidate="txtUserName" Text="用户名必填">
        </asp:RequiredFieldValidator></td>
    </tr>
    <tr>
        <td align="right">密码：</td>
        <td><asp:TextBox ID="txtUserPass" runat="server" TextMode=
         "Password" CssClass="frame-gray1"></asp:TextBox><asp:Required
         FieldValidator ID="rfvPass" runat="server" ControlToValidate=
         "txtUserPass" Text="密码必填"></asp:RequiredFieldValidator></td>
    </tr>
    <tr>
        <td align="right">确认密码：</td>
        <td><asp:TextBox ID="txtUserPassAgain" runat="server" TextMode=
        "Password" CssClass="frame-gray1"></asp:TextBox><asp:RequiredField
        Validator ID="rfvPassAgain" runat="server" ControlToValidate=
        "txtUserPassAgain" Text="确认密码必填"></asp:RequiredFieldValidator>
        <asp:CompareValidator ID="cvPassAgain" runat="server" ControlTo
        Validate ="txtUserPassAgain" ControlToCompare="txtUserPass" Text
        ="两次密码不一致"></asp:CompareValidator></td>
    </tr>
    <tr>
```

```
            <td align="right">您的邮箱：</td>
            <td><asp:TextBox ID="txtUserMail" runat="server" TextMode="Email"
            CssClass="frame-gray1"></asp:TextBox><asp:RequiredFieldValidator
            ID="rvfEmail" runat="server" ControlToValidate="txtUserMail"
            Text="邮箱必填"></asp:RequiredFieldValidator></td>
        </tr>
        <tr>
            <td></td>
            <td><asp:CheckBox ID="cbTrue" runat="server" />我已阅读 <a href="#"
            target="_blank">用户注册协议</a></td>
        </tr>
        <tr>
            <td colspan="2" align="left"><p class="checkboxP"><asp:Button
           ID="btnRegister" runat="server" CssClass="register" BorderStyle
          ="None" /></p></td>
        </tr>
    </table>
```


在上述代码中，分别在【用户名】、【密码】、【确认密码】和【您的邮箱】输入框之后添加 RequiredFieldValidator 控件，指定这些内容必须填写。同时还为【确认密码】输入框指定 CompareValidator 控件，设置该控件 ControlToCompare 属性的值为 txtUserPass，它是【密码】输入框的 ID 属性值。

运行上述代码直接单击按钮进行测试，如图 8-3 所示。在【密码】输入框和【确认密码】输入框中输入内容，如图 8-4 所示。

图 8-3　注册页面

图 8-4　判断密码

从图 8-4 中可以看出，在【密码】框和【确认密码】框中输入内容后单击按钮时，提示“两次密码不一致”。但是，为什么在提示左侧会有空白呢？这是验证同时为确认密码框添加必填验证和比较验证，虽然必填验证已经通过，但是，此控件的 Display 属性的值为 Static，因此，它还占据着一定的空间。如果不想显示多余的空白，可以将该控件 Display 属性的值设置为 Dynamic。代码如下。

```
<asp:RequiredFieldValidator ID="rfvPassAgain" runat="server" Control
ToValidate="txtUserPassAgain" Display="Dynamic" Text="确认密码必填">
</asp:RequiredFieldValidator>
```

重新运行页面查看效果，如图 8-5 所示。



图 8-5 **Display 属性的值为 Dynamic**

2. 控件值与指定值比较

开发者还可以将控件的值与指定的常用值进行比较，如范例 4 所示。

【范例 4】

大家经常会玩猜数字游戏，一方出数字，另一方猜。如果猜错，就一直让对方猜，直到猜对为止。在页面中添加一个 TextBox 控件，然后分别添加 RequiredFieldValidator 控件和 CompareValidator 控件，最后添加 Button 控件和 Label 控件。代码如下。

```
输入一个 10 以内的数字：<asp:TextBox ID="txtNumber" runat="server">
</asp:TextBox>
<asp:RequiredFieldValidator ID="rvfNumber" runat="server" Text="请输入数
字" Display="Dynamic" ControlToValidate="txtNumber"></asp:RequiredField
Validator>
<asp:CompareValidator ID="cvNumber" runat="server" Text="您猜的数字错误"
Display="Dynamic" ControlToValidate="txtNumber"></asp:CompareValidator>
<asp:Button ID="btnNumber" runat="server" Text=" 测 试 " OnClick=
"btnNumber Click" /><br /><br />
<asp:Label ID="lblResult" runat="server"></asp:Label>
```

在上述代码中，RequiredFieldValidator 控件验证 TextBox 控件的值不能为空，CompareValidator 控件将用户输入的值与指定的随机数值进行比较，Button 控件执行操作，Label 控件显示执行正确时的结果。

在后台中为页面的 Load 事件添加代码，首次加载时获取 10 以内的随机数字，并将其指定为 CompareValidator 控件的 ValueToCompare 属性值。代码如下。

```
protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack) {
        Random dom = new Random();
        int number = dom.Next(10);
        cvNumber.ValueToCompare = number.ToString();
    }
}
```

在后台为 Button 控件添加 Click 事件，代码如下。

```
protected void btnNumber_Click(object sender, EventArgs e) {
    if (IsValid) {
        lblResult.Text = "恭喜您，这个数字猜对了呢";
    }
}
```

运行页面输入数字进行测试，如图 8-6 和图 8-7 所示分别为测试失败和成功时的效果。

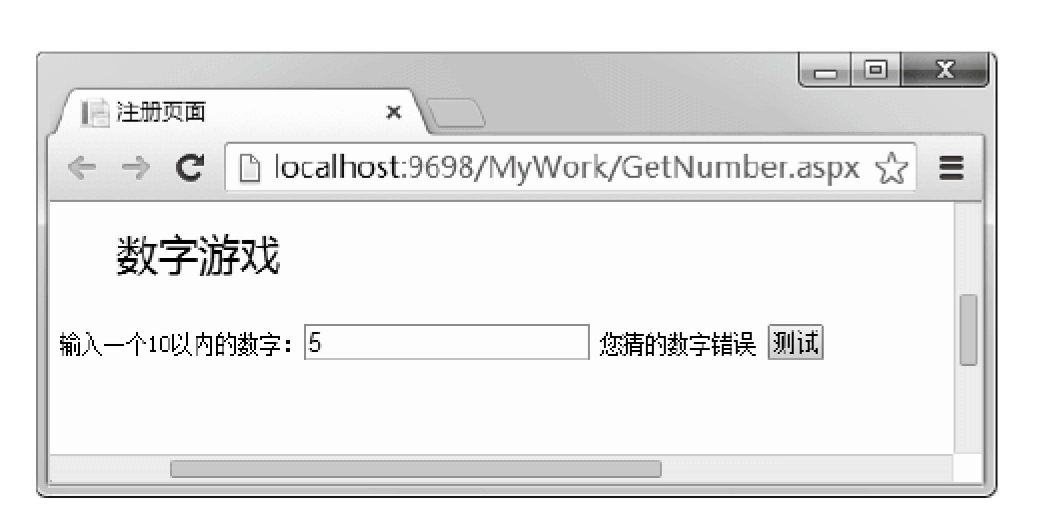


图 8-6 测试失败

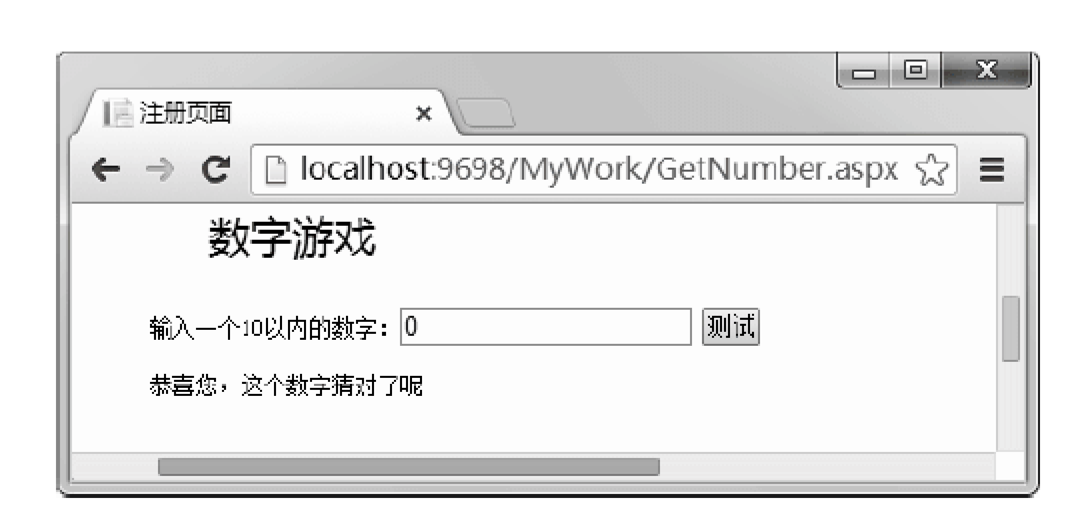


图 8-7 测试成功

8.2.3 范围验证控件

在 ASP.NET 中，RangeValidator 表示范围验证控件。范围验证控件用于验证输入控件的值是否在指定的范围内。如某些公司在招聘员工时，会要求员工年龄在 20～35 岁之间，这就可以通过 RangeValidator 控件实现。

RangeValidator 控件的大多数属性与 RequiredFieldValidator 控件类似，如 ControlToValidate、Display 和 Text 等。下面列出了 RangeValidator 专用的三个常用属性。

（1）MinimumValue 属性：获取或设置验证范围的最小值。

（2）MaximumValue 属性：获取或设置验证范围的最大值。

（3）Type 属性：获取或设置在比较之间将所比较的值转换到的数据类型。Type 属性的值是枚举类型 ValidationDateType 的值之一，其值说明如下。

① String：默认值，字符串数据类型，它的值被看作是 System.String。

② Date：日期数据类型，只允许使用数字日期，不能指定时间部分。

③ Integer：32 位符号整数数据类型，它的值被看作是 System.Int32。

④ Double：双精度浮点数数据类型，它的值被看作是 System.Double。

⑤ Currency：货币数据类型，它的值被看作是 System.Decimal。

【范例 5】

在页面中添加【请输入年龄】输入框和【测试】按钮，指定用户年龄必须输入，并且它的值在 15～40 岁之间。通过 RequiredFieldValidator 控件指定必填，RangeValidator 控件指定年龄的范围和类型。代码如下。

```
请输入年龄：<asp:TextBox ID="txtAge" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvAge" runat="server" ControlTo
Validate="txtAge" Text="必须输入" Display="Dynamic"></asp:
RequiredFieldValidator>
<asp:RangeValidator ID="rvAge" runat="server" ControlToValidate=
"txtAge" Type="Integer" MaximumValue="40" MinimumValue="15" Display=
"Dynamic" Text="年龄必须在 15 岁到 40 岁之间"></asp:RangeValidator>
<asp:Button ID="btnAge" runat="server" Text="测试" />
```

8.2.4 正则表达式验证控件

在 ASP.NET 中，RegularExpressionValidator 表示正则表达式验证控件。正则表达式控件用于验证指定用户输入的内容是否与某个正则表达式所定义的模式相匹配。这类验证允许用户检查可预知的字符序列，例如身份证号、电子邮件地址、电话号码和邮政编码中的字符序列等。

RegularExpressionValidator 控件的大多数属性与 RequiredFieldValidator 相似，但是它还特有一个 ValidationExpression 属性。ValidationExpression 属性用于获取或设置确定字段验证模式的正则表达式。

开发者在设置 ValidationExpression 属性时，可以使用提供的一些正则表达式。选择 RegularExpressionValidator 控件右击，选择【属性】命令打开【属性】窗格，在【属性】窗格中打开 ValidationExpression 属性，单击该属性后面的按钮，弹出如图 8-8 所示的【正则表达式编辑器】对话框。



图 8-8 【正则表达式编辑器】对话框

在如图 8-8 所示对话框中，选择正则表达式后单击【确定】按钮即可。如果想自定义，可以选择图中的（Custom）选项，然后在【验证表达式】输入框中输入内容，再单击【确定】按钮即可。如下代码为一些常用的正则表达式。

```
^[0-9]*$                    //验证数字
^\d{n}$                     //验证 n 位的数字
^[A-Za-z]+$                 //验证由 26 个英文字母组成的字符串
^[\u4e00-\u9fa5A-Za-z0-9-_]*$//验证用户名或昵称只能是英文、数字、下划线和减号
^[A-Za-z0-9]+$              //验证由数字和 26 个英文字母组成的字符串
```

```
^[\u4e00-\u9fa5],{0,}$          //验证汉字
^(0?[1-9]|1[0-2])$      //验证一年的 12 个月：正确格式为："01"-"09"和"1""12"
^((0?[1-9])|((1|2)[0-9])|30|31)$
                         //验证一个月的 31 天：正确格式为：01、09 和 1、31
(86)*0*13\d{9}                   //中国手机号码
(\(\d{3,4}\)|\d{3,4}-|\s)?\d{8}          //中国固定手机号码
(\(\d{3,4}\)|\d{3,4}-|\s)?\d{7,14}      //中国电话号码（包括移动和固定电话）
```

无论是 ValidationExpression 属性提供的正则表达式，还是自定义的正则表达式，不同的字符代表不同的含义，常用的字符说明如下所示。

（1）.：匹配除\n 之外的任意单个字符。

（2）^：匹配输入的开始位置。

（3）$：匹配输入的结尾。

（4）*：匹配前面的子表达式零次或多次（大于等于 0 次），等价于{0,}。

（5）+：匹配前一个字符一次或多次。例如，"zo+" 可以匹配 "zoo"，但不匹配 "z"。

（6）?：匹配前一个字符零次或一次。例如，"a?ve?" 可以匹配 "never" 中的 "ve"。

（7）[A-Z]：表示任意大写字母。

（8）\d：表示任意一个数字。

（9）\w：匹配包括下划线的任意单词字符，等价于[A-Za-z0-9_]。

【范例 6】

在页面中添加不同的输入框和 RegularExpressionValidator 控件，这些控件分别验证身份证号、固定电话、某年中的月份。代码如下。

```
<div>
    请输入身份证号：<asp:TextBox ID="txtCardNo" runat="server"></asp:
    TextBox>
    <asp:RegularExpressionValidator ID="revCardNo" runat="server" Control
    ToValidate="txtCardNo" Display="Dynamic" Text="输入的身份证号有误"
    ValidationExpression="\d{17}[\d|X]|\d{15}">输入的身份证号有误</asp:
    Regular ExpressionValidator>
</div>
<div>
    请输入固定电话：<asp:TextBox ID="txtPhone" runat="server"></asp:TextBox>
    <asp:RegularExpressionValidator ID="revPhone" runat="server" Control
    ToValidate="txtPhone" Display="Dynamic" Text="输入的电话号码有误，格式
    为 XXX-XXXXXXXX 或者 XXXX-XXXXXXXX" ValidationExpression="(\(\d{3,4}\)
    |\d{3,4}-|\s)?\d{8}"></asp:RegularExpressionValidator>
</div>
<div>
    请输入某年月份：<asp:TextBox ID="txtMonth" runat="server"></asp:Text
    Box>
    <asp:RegularExpressionValidator ID="revMonth" runat="server" ControlTo
    Validate="txtMonth" Display="Dynamic" ValidationExpression="^(0?
    [1-9]|1[0-2])$" Text="月份有错误，格式为：01"></asp:RegularExpression
```

```
    Validator>
</div>
<div style="margin-top: 20px"><asp:Button ID="btnTest" runat="server"
Text=" 测 试 " /></div>
```

运行页面输入内容后单击按钮进行测试，如图 8-9 所示。

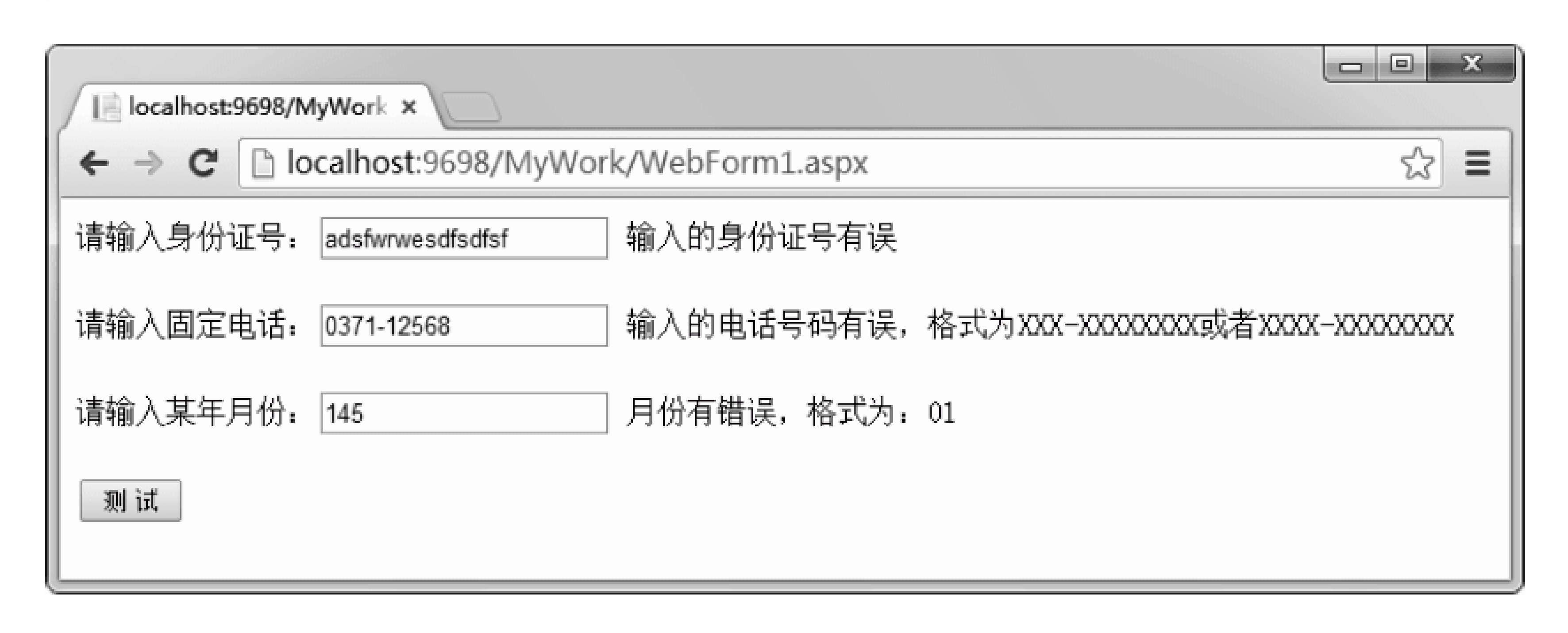


图 8-9　正则表达式验证控件

8.2.5　自定义验证控件

如果开发者不想使用上述介绍的验证控件，那么还可以使用自定义验证控件。在 ASP.NET 中，CustomValidator 表示自定义验证控件。自定义验证控件使开发者使用自己编写的验证逻辑检查用户的输入。

CustomValidator 控件的多数常用属性可以参考 RequiredFieldValidator 控件，其常用的特有属性如表 8-3 所示。

表 8-3　CustomValidator 控制的特有属性

属性名称	说明
ClientValidationFunction	用户设置客户端验证的脚本函数
ValidateEmptyText	获取或设置一个值，该值指示是否验证空文本。默认值为 false
OnServerValidate	服务器端验证的事件方法
EnableClientScript	指示是否在上级浏览器中对客户端执行验证

1. 自定义客户端验证

CustomValidator 控件既支持客户端脚本验证，又支持服务器端验证。在实现客户端验证时，需要在网页中定义脚本函数。函数原型如下：

```
function ValidationFunctionName(source, arguments)
```

其中，ValidationFunctionName 表示函数的名称，需要向函数中传入两个参数，source 是对 CustomValidator 控件呈现的元素的引用，arguments 是一个具有 Value 属性和 IsValid 属性的对象，该参数可以获取控件的值。

定义脚本函数完毕后，还需要设置 CustomValidator 控件的 ClientValidationFunction

属性，它指定客户端验证的脚本函数。

【范例 7】

分别向两个输入框中输入数字，然后计算这两个数字之间所有数字的和，必须保证用户输入的第二个数字大于第一个数字。例如，如果输入第一个数字 1，输入第二个数字 5，那么将计算 1+2+3+4+5 的结果。步骤如下。

（1）创建 Result.aspx 页面，在页面的表单元素中添加四行两列的表格。其中，第一行表示输入的起始数字。代码如下。

```
<asp:TextBox  ID="txtFirst"  runat="server"  CssClass="frame-gray1">
</asp:TextBox>
```

（2）第二行表示输入的结尾数字，该数字必须比起始数字大。在 TextBox 控件后添加 CustomValidator 控件。代码如下。

```
<asp:TextBox  ID="txtSecond"  runat="server"  CssClass="frame-gray1">
</asp:TextBox>
<asp:CustomValidator  ID="cvTest"  runat="server"  ControlToValidate=
"txtSecond"  Text=" 必 须 比 第 一 个 数 字 大 "  ClientValidationFunction=
"NumberTest"></asp:CustomValidator>
```


（3）添加执行操作的 Button 控件和显示结果的 Label 控件，并为 Button 控件添加 Click 事件。代码如下。

```
<asp:Button  ID="btnResult"  runat="server"  Text=" 计 算 结 果 "  OnClick
="btnResult Click" />
<asp:Label ID="lblMessage" runat="server"></asp:Label>
```

（4）创建名称为 NumberTest 的脚本函数，在该函数中获取起始数字和结尾数字，通过 isNaN()函数判断输入的内容是否为数字。如果不是弹出提示，否则判断结尾数字是否小于等于起始数字，如果是则 IsValid 属性的值设置为 false，否则为 true。代码如下。

```
function NumberTest(obj, args) {
    var first = document.getElementById("txtFirst").value;
    var second = document.getElementById("txtSecond").value;
    if (isNaN(first) || isNaN(second))   //如果不为数字
        alert("文本框内必须是数字。如 1、2、3、4");
    else {
        if (second <= first) {
            args.IsValid = false;
        } else {
            args.IsValid = true;
        }
    }
}
```

（5）为页面中的 Button 控件添加 Click 事件代码，在事件代码中判断 IsValid 属性的值是否通过验证，如果通过验证则通过 for 语句计算结果，并将结果显示到页面。代码

如下。

```
protected void btnResult Click(object sender, EventArgs e) {
    if (IsValid) {      //如果通过验证
        int result = 0;
        for (int i = Convert.ToInt32(txtFirst.Text); i <= Convert.ToInt
        32(txtSecond.Text); i++) {
            result += i;
        }
        lblMessage.Text = "从" + txtFirst.Text + "到" + txtSecond.Text + "
         相加的数字和是：" + result;
    }
}
```

（6）运行页面输入内容后进行测试，如图 8-10 所示为结尾数字小于起始数字时的效果，图 8-11 为计算后的结果。



图 8-10　结尾数字小于起始数字

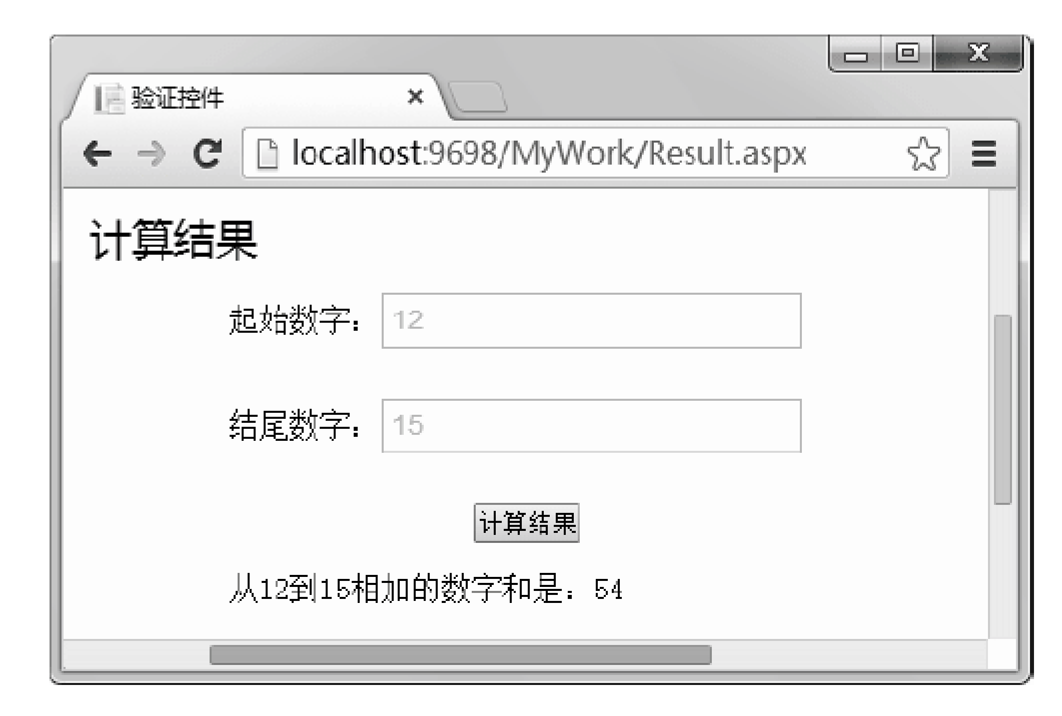


图 8-11　将数字循环相加后的结果

2．自定义服务器端验证

通过 CustomValidator 控件实现服务器端验证时，需要在该控件的 ServerValidate 事件中添加代码。ServerValidate 事件的原型如下。

```
protected  void  CustomValidator1_ServerValidate(object  source,  Server
ValidateEventArgs args) {
    //代码
}
```

其中，source 是对引发 ServerValidate 事件的自定义验证控件的引用；args 是 ServerValidateEventArgs 对象，通过该对象的 Value 属性获取用户输入的内容，如果该内容有效，则 args.IsValid 的值为 true，否则为 false。

【范例 8】

更改范例 7 的代码，为 CustomValidator 控件添加 ServerValidate 事件实现服务器端验证。首先删除 CustomValidator 控件的 ClientValidationFunction 属性和 NumberTest()函数，为其添加 ServerValidate 事件。页面代码如下。

```
<asp:CustomValidator ID="cvTest" runat="server" ControlToValidate=
"txtSecond" Text="必须比第一个数字大" OnServerValidate="cvTest_Server
Validate"></asp:CustomValidator>
```

然后在后台页面添加 CustomValidator 控件的 ServerValidate 事件，代码如下。

```
protected void cvTest_ServerValidate(object source, ServerValidateE
ventArgs args) {
    int num1 = 0, num2 = 1;
    bool ret = true;
    try {
        num1 = Convert.ToInt32(txtFirst.Text);
        num2 = Convert.ToInt32(txtSecond.Text);
    } catch (Exception ex) {
        args.IsValid = false;
        ret = false;
        lblMessage.Text = "出现错误，原因是：" + ex.Message;
    }
    if (ret) {
        if (num2 <= num1) {
            args.IsValid = false;
        } else {
            args.IsValid = true;
        }
    }
}
```


运行页面输入内容进行测试，效果图不再显示。

8.3 错误验证汇总控件

在 ASP.NET 中提供的 6 种控件中，8.2 节介绍的 5 个控件都是基础验证控件，最后一个控件是 ValidationSummary 控件，它是错误验证汇总控件。该控件用于在一个位置总结来自网页中所有验证程序的错误信息，该控件可以将错误信息归纳在一个简单的列表中，以内联方式或者摘要方式显示错误。

ValidationSummary 控件提供多个属性，常用属性如表 8-4 所示。

表 8-4 ValidationSummary 控件的常用属性

属性名称	说明
DisplayMode	获取或设置验证摘要的显示模式，取值说明如下。 （1）BulletedList：默认值，显示在项目符号中的验证摘要 （2）List：显示在列表中的验证摘要 （3）SingleParagraph：显示在单个段落内的验证摘要
EnableClientScript	获取或设置一个值，用于指示该控件是否使用脚本更新自己
HeaderText	获取或设置显示在摘要上方的标题文本
ShowMessageBox	获取或设置一个值，该值指示是否在消息框中显示摘要信息
ShowSummary	获取或设置一个值，该值指示是否内联显示验证摘要

【范例 9】

在范例 6 的基础中更改和添加新的代码，实现步骤如下。

（1）将 RegularExpressionValidator 控件的 Display 属性的值设置为 None，并为该控件添加 ErrorMessage 属性。部分代码如下。

```
请输入身份证号：<asp:TextBox ID="txtCardNo" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator ID="revCardNo" runat="server" ControlTo
Validate="txtCardNo" Display="None" ErrorMessage="身份证号错误" Text="输
入的身份证号有误" ValidationExpression="\d{17}[\d|X]|\d{15}">输入的身份证号
有误</asp:RegularExpressionValidator>
```

（2）添加 ValidationSummary 控件，指定控件的 HeaderText 属性。代码如下。

```
<asp:ValidationSummary ID="vsTotal" runat="server" HeaderText="验证错误
信息" />
```

（3）运行页面输入内容查看效果，如图 8-12 所示。

（4）为 ValidationSummary 控件添加 DisplayMode 属性，将属性值设置为 List。然后重新运行页面进行测试，如图 8-13 所示。

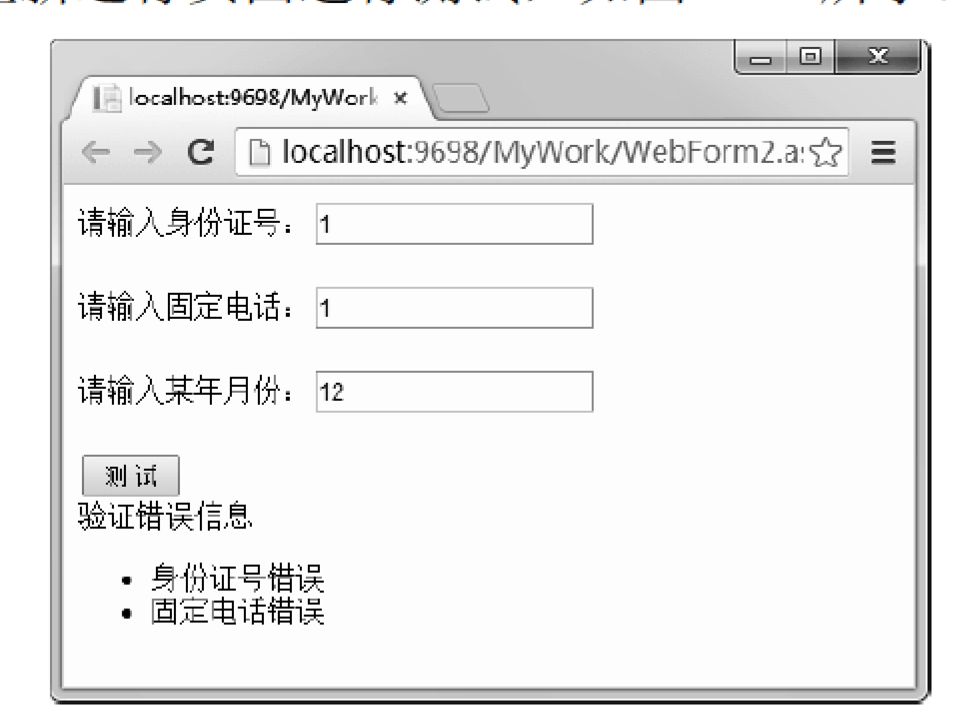


图 8-12 项目符号验证摘要



图 8-13 列表中的验证摘要

（5）更改 DisplayMode 属性的值，将其指定为 SingleParagraph。然后重新运行页面进行测试，如图 8-14 所示。

（6）将 ValidationSummary 控件 ShowMessageBox 属性的值设置为 true。然后重新运行页面进行测试，如图 8-15 所示。

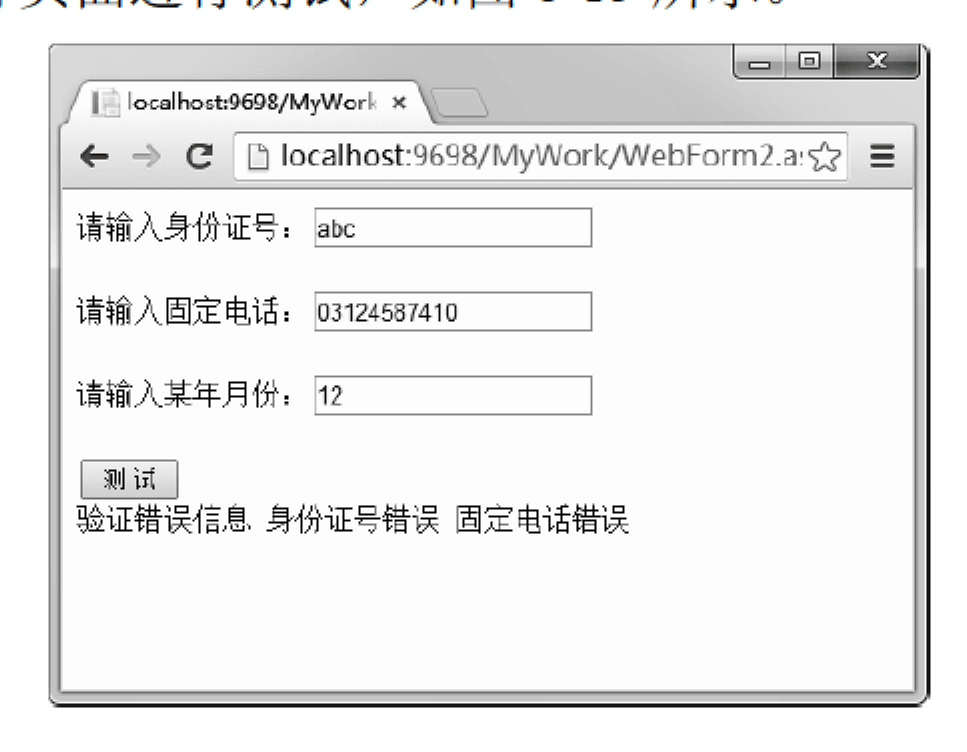


图 8-14 在单个段落内验证摘要

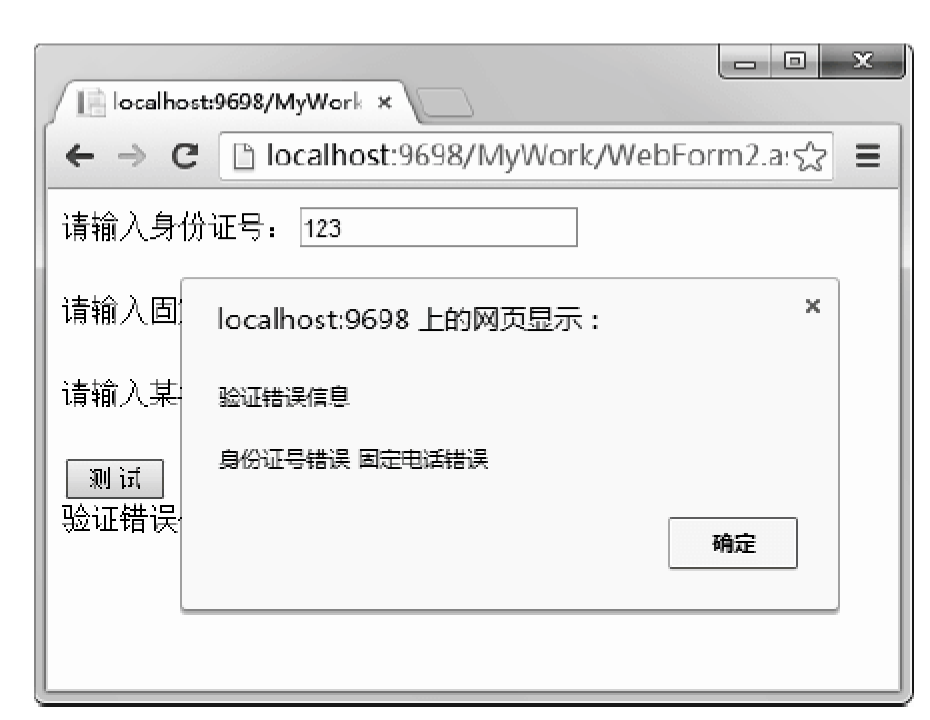


图 8-15 ShowMessageBox 属性

ValidationSummary 控件能够以消息框的形式显示摘要信息，也能以内联的形式（即在网页中输出）显示摘要信息。以消息框的形式显示摘要需要将 ShowMessageBox 属性的值设置为 true，ShowSummary 属性的值设置为 false；以内联方式的形式显示摘要需要将 ShowSummary 属性的值设为 true，ShowMessageBox 属性的值设置为 false。如果同时设置 ShowMessageBox 属性和 ShowSummary 属性，这时两种形式都会显示。

8.4 实验指导——ValidationGroup 属性实现分组验证

在一个网页中可能出现多个按钮，这些按钮执行不同的操作。假设在当前的页面包含两部分内容：第一部分提供用户输入登录信息，包括登录名和登录密码；第二部分提供用户输入注册信息，包括用户名、密码、确认密码、电子邮箱、手机号码、出生日期以及现住地址等。每一部分都包含一个操作按钮，单击不同的按钮执行不同的操作，这需要借助于 ValidationGroup 属性。

ValidationGroup 属性指定当回发事件生成时要验证的验证组，只验证指定验证组中的验证控件。

本节综合利用 ValidationGroup 属性和验证控件等知识实现分组验证的功能。步骤如下。

（1）创建 ValidationGroup.aspx 页面并进行设计，首先创建三行两列的表格，第一行提供登录名，第二行提供登录密码，第三行提供操作按钮。相关代码如下。

```
<table width="100%" height="200">
   <tr>
      <td align="right">登录名：</td>
      <td><asp:TextBox ID="txtLoginName" runat="server" CssClass=
      "frame-gray1"></asp:TextBox><asp:RequiredFieldValidator  ID="rfv
      LoginName" runat="server" ControlToValidate="txtLoginName" Display
     ="Dynamic" Text="登录名必须输入" ForeColor="Blue" Font-Size="14">
      </asp:RequiredFieldValidator></td>
   </tr>
   <tr>
      <td align="right">登录密码：</td>
      <td><asp:TextBox ID="txtLoginPass" TextMode="Password" runat=
      "server" CssClass="frame-gray1"></asp:TextBox><asp:RequiredField
      Validator ID="rvfLoginPass" runat="server" ControlToValidate="txt
      LoginPass" Display="Dynamic" Text="登录密码必须输入" ForeColor=
      "Blue" Font-Size="14"></asp:RequiredFieldValidator></td>
   </tr>
   <tr>
      <td></td>
      <td><asp:Button ID="btnLogin" runat="server" Text="登录" Width=
      "90" Height="30" BackColor="#446af0" ForeColor="White" Font-Size=
      "16" /></td>
```

```
    </tr>
</table>
```

（2）继续创建七行两列的表格，第一行提供【用户名】输入框，该输入框中必须填写内容。代码如下。

```
<asp:TextBox ID="txtName" runat="server" CssClass="frame-gray1"></asp:
TextBox>
<asp:RequiredFieldValidator  ID="rfvName"  runat="server"  ControlTo
Validate="txtName" Display="None" ErrorMessage="用户名必须输入"></asp:
RequiredFieldValidator>
```

（3）第二行和第三行提供【密码】输入框和【确认密码】框，指定【密码】框必须填写，【确认密码】框的内容与【密码】框的内容一致。相关代码如下。

```
<asp:TextBox ID="txtPass" TextMode="Password" runat="server" CssClass
="frame-gray1"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvPass" runat="server" ControlToValidate
="txtPass" Display="None" ErrorMessage="密码必须输入"></asp:Required
FieldValidator>
<asp:TextBox ID="txtPassAgain" TextMode="Password" runat="server" CssClass
="frame-gray1"></asp:TextBox>
<asp:CompareValidator ID="cvPass" runat="server" Display="None" ControlTo
Validate="txtPassAgain" ErrorMessage="两次密码不一致" ControlToCompare
="txtPass"></asp:CompareValidator>
```

（4）第四行提供用户输入出生日期，指定 TextBox 控件的 TextMode 属性值为 Date。相关代码如下。

```
<asp:TextBox  ID="txtBirth"  runat="server"  TextMode="Date"  CssClass=
"frame-gray1"></asp:TextBox>
<asp:RangeValidator  ID="rvBirth"  runat="server"  ControlToValidate=
"txtBirth" Display="None" MinimumValue="1900-01-01" MaximumValue="9999-
12-12" ErrorMessage="输入的日期不合法"></asp:RangeValidator>
```

（5）第五行提供用户输入的电子邮箱，电子邮箱必须填写，并且通过 RegularExpressionValidator 控件验证输入的邮箱是否正确。代码如下。

```
<asp:TextBox ID="txtMail" runat="server" CssClass="frame-gray1"></asp:
TextBox>
<asp:RequiredFieldValidator  ID="rvfMail"  runat="server"  ControlTo
Validate="txtMail" ErrorMessage="电子邮箱必须输入" Display="None">
</asp:RequiredFieldValidator>
<asp:RegularExpressionValidator ID="revMail" runat="server" ControlTo
Validate="txtMail" ErrorMessage="电子邮箱不合法" Display="None" Validation
Expression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"></asp:Regular
ExpressionValidator>
```

（6）第六行提供用户执行的操作按钮，代码如下。

```
<asp:Button ID="btnRegister" runat="server" Text="注册" Width="90"
Height="30" BackColor="#446af0" ForeColor="White" Font-Size="16" />
```

（7）第七行提供一个 ValidationSummary 控件，它通过消息框的形式弹出错误信息。代码如下。

```
<asp:ValidationSummary ID="vsTotal" runat="server" ShowMessageBox=
"true" ShowSummary="false" />
```

（8）运行页面查看效果，单击页面中的一个按钮进行测试，如图 8-16 所示。



图 8-16　验证效果 1

从上述效果图中可以明显看出，当单击图中的【登录】按钮时，不仅显示与用户登录输入框有关的提示，还会弹出与用户注册输入框有关的消息框。这与开发者要实现的功能不一致。这还需要最关键的一步，即设置 ValidationGroup 控件的属性。

（9）在用户登录相关的验证控件和按钮中添加 ValidationGroup 属性，指定 ValidationGroup 属性的值为 LoginValidator。相关代码如下。

```
<asp:RequiredFieldValidator ID="rfvLoginName" runat="server" Validation
Group="LoginValidation" ControlToValidate="txtLoginName" Display="Dynamic"
Text="登录名必须输入" ForeColor="Blue" Font-Size="14"></asp:Required
FieldValidator>
<asp:RequiredFieldValidator ID="rvfLoginPass" runat="server" Validation
Group="LoginValidation" ControlToValidate="txtLoginPass" Display="Dynamic"
Text="登录密码必须输入" ForeColor="Blue" Font-Size="14"></asp:Required
FieldValidator>
```

```
<asp:Button ID="btnLogin" runat="server" Text="登录" ValidationGroup=
"LoginValidation" Width="90" Height="30" BackColor="#446af0" ForeColor=
"White" Font-Size="16" />
```

（10）在用户注册相关的验证控件和按钮中添加 ValidationGroup 属性，指定 ValidationGroup 属性的值为 RegisterValidatior。以 Button 控件和 ValidationSummary 控件为例，相关代码如下。

```
<asp:Button ID="btnRegister" runat="server" Text="注册" Validation
Group="RegisterValidation" Width="90" Height="30" BackColor="#446af0"
ForeColor="White" Font-Size="16" />
<asp:ValidationSummary ID="vsTotal" runat="server" ValidationGroup=
"RegisterValidation" ShowMessageBox="true" ShowSummary="false" />
```

（11）直接刷新页面或重新运行页面，单击页面中的【注册】按钮进行测试，如图 8-17 所示。



图 8-17 验证效果 2

思考与练习

一、填空题

1．ASP.NET 中提供的__________控件用于确保用户不会跳过某一项输入。

2．验证控件的 Display 属性的可取值包括 Static、__________和 None。

3．RangeValidator 控件的__________属性可

以设置验证范围的最大值。

4. RegularExpressionValidator 控件的______属性用于获取或设置确定字段验证模式的正则表达式。

5．CustomValidator 控件实现服务器端验证时，需要指定该控件的__________事件。

6．__________属性指定当回发事件生成时要验证的验证组。

二、选择题

1．对于验证控件来说，__________属性是必须进行设置的。

A．ControlToValidate

B．ControlToCompare

C．Display

D．ErrorMessage

2．CompareValidator 控件的 Type 属性的默认取值是__________。

A．Date

B．Currency

C．Integer

D．String

3．假设出生日期输入框的值不能为空，且出生日期的值必须在 1900 年 1 月 1 日到 2013 年 12 月 12 日之间。那么开发者在验证过程中，最不可能使用到__________控件。

A．CustomValidator

B．CompareValidator

C．RangeValidator

D．RequiredFieldValidator

4．自定义验证控件实现数据验证时，__________属性设置客户端验证的脚本函数。

A．ValidateEmptyText

B．ClientValidationFunction

C．OnServerValidate

D．EnableClientScript

5．ValidationSummary 控件的 DisplayMode 属性的默认值是__________。

A．Circle

B．SingleParagraph

C．List

D．BulletedList

6. 将 ValidationSummary 控件的__________属性的值设置为 true 时，表示以消息框的形式显示摘要。

A．ShowHeaderList

B．ShowBox

C．ShowSummary

D．ShowMessageBox

三、简答题

1. 在 ASP.NET 中进行数据验证的方法有哪些？请简单进行说明。

2．分别概述 ASP.NET 中提供的 6 种验证控件。

3．ValidationGroup 属性是干什么的？如何使用 ValidationGroup 属性？

第 9 章　ADO.NET 进行数据库编程

ADO.NET 是.NET 平台的数据访问技术体系，这是微软从 COM 时代奠基的 OLE DB 技术发展而来的。在 OLE DB 之上建立了一个很好的数据存取模型 ADO，并被业界接受，如 VC 6.0、Delphi5/6 等都使用 ADO 数据存取技术。ADO.NET 继承了 ADO 的优点，但它是微软在.NET 平台下采用全新的架构和理念构建的。ADO.NET 主要通过 Connection、Command、Parameter 和 DataReader 等对象实现对数据库的访问和操作。

ADO.NET 经过不断发展，已经成为包含 LINQ、Entity Framework 等新一代 ORM（Object-Relation Mapping，对象-关系映射）技术的完整体系。就像用 LINQ 和 Entity Framework 来指代新的技术一样，开发者还是习惯使用 ADO.NET 一词代表最初的 ADO.NET 技术。本章简单介绍常用的 Connection、Command 和 DataAdapter 等对象，通过这些对象进行数据库编程。

本章学习要点：

- ❑ 熟悉 ADO.NET 的组成
- ❑ 掌握 SqlConnection 对象的使用
- ❑ 掌握 SqlCommand 对象的使用
- ❑ 掌握 SqlParameter 对象的使用
- ❑ 掌握 SqlDataReader 对象的使用
- ❑ 熟悉 SqlDataReader 和 DataSet 的区别
- ❑ 熟悉 DataSet 对象的使用
- ❑ 了解 SqlDataAdapter 对象的使用
- ❑ 掌握 DataTable 对象的创建和使用
- ❑ 了解 DataView 对象的创建和使用

9.1　ADO.NET 概述

ADO.NET 技术是一组面向.NET Framework 提供数据访问服务的类，它为创建分布式数据共享应用程序提供了一组丰富的组件。ADO.NET 不仅适用于对关系型数据库系统的访问，还支持对文本文件、Excel 表格 XML 和应用程序数据的访问，是.NET Framework 中不可缺少的一部分。

ADO.NET 支持多种开发需求，包括创建由应用程序、工具、语言或 IE 浏览器使用的前端数据库客户端和中间层业务对象。ADO.NET 有多个特点，主要特点如下。

（1）ADO.NET 通过数据处理将数据访问分解为多个可以单独使用或一前一后使用的不连续组件，它包含用于连接到数据库、执行命令和检索结果的.NET Framework 数据提供程序。

（2）ADO.NET 类位于 System.Data.dll 中，并与 System.Xml.dll 中的 XML 集成。

（3）ADO.NET 向编写托管代码的开发者提供类似于 ActiveX 数据对象向本机组件对象模型开发者提供的功能。

（4）ADO.NET 在.NET Framework 中提供最直接的数据访问方法。

ADO.NET 包含.NET Framework 数据提供程序和 DataSet 两个组件。.NET Framework 数据提供程序是专门为数据操作以及快速、只进、只读访问数据而设计的组件。如表 9-1 所示列出了 4 种类型的数据提供程序。

表 9-1 .NET Framework 数据提供程序

数据提供程序名称	说明
SQL Server .NET Framework 数据提供程序	适合 SQL Server 公开的数据源，使用 System.Data.SqlClient 命名空间
OLE DB .NET Framework 数据提供程序	适合 OLE DB 公开的数据源，使用 System.Data.OleDb 命名空间
ODBC .NET Framework 数据提供程序	适合 ODBC 公开的数据源，使用 System.Data.Odbc 命名空间
Oracle .NET Framework 数据提供程序	适合 Oracle 公开的数据源，使用 System.Data.Oracle 命名空间

数据提供程序提供包含访问各种数据源数据的对象，如 Connection、Command、DataReader、DataAdapter、Parameters 等。

在实际使用过程中，具体使用哪种应用程序是由用户使用的数据库决定的。在如表 9-1 所示的 4 种数据提供程序中，SQL Server .NET Framework 数据提供程序最为常用。因此，本书主要使用 SQL Server 数据库对数据进行操作。

ADO.NET 专门提供了一套用于访问数据库的对象，这些对象都存在于 System.Data.SqlClient 命名空间下，并且需要对上面提到的对象添加前缀。对于 SQL Server 数据库来说，使用 Connection 和 Command 这些对象时需要添加 Sql 前缀，如 SqlConnection 对象、SqlCommand 对象和 SqlDataReader 对象等，下面详细介绍这些对象。

DataSet 是专门为独立于任何数据源的数据访问而设计的，它可以用于多种不同的数据源、XML 数据或者管理程序本地的数据。简单来说，DataSet 对象可以在断开数据库连接的情况下访问数据。

9.2 SqlConnection 对象

要访问或者操作后台数据库中的数据，必须先连接到数据库。SqlConnection 是 SQL Server 数据库连接对象，该对象提供对 SQL Server 数据库的连接，但是并不能对数据库发送任何的 SQL 命令。

9.2.1 创建 SqlConnection 对象

使用 SqlConnection 对象完成连接数据库的操作过程如下。

（1）创建 SqlConnection 对象。
（2）打开数据库连接。
（3）访问或者操作后台数据库中的数据。
（4）数据处理完毕后，关闭数据库连接。

SqlConnection 是对象，就像 C#中其他的对象一样。很多时候，开发者只需要声明并实例化。语法如下。

```
SqlConnection conn = new SqlConnection(string connectionString);
```

上面实例化 SqlConnection 对象是使用一个带 string 类型参数的构造函数，这个参数称为连接字符串。如表 9-2 所示描述了连接字符串的常用部分。

表 9-2 连接字符串的常用部分

连接字符串参数名	说明
Data Source	数据源，一般为机器名或 IP 地址。如果是本机，可以使用点（.）代替，也可以使用 localhost
Initial Catalog	数据库或 SQL Server 实例的名称（与 Database 一样）
Integrated Security	设置为 SSPI，使连接使用用户的 Windows 登录
User ID（Uid）	登录数据库时的用户名称
Password（Pwd）	登录数据库的用户密码。如果密码为空，则该项可以省略
Database	数据库或 SQL Server 实例的名称
Server	数据库所在的服务器名称，一般为机器名称
Pooling	表示是否启用连接池。如果为 true 则表示启用连接池
Connection Timeout	连接超时时间，默认值为 15s

【范例 1】

首先声明一个连接字符串，通过该字符串指定数据源，数据库，登录数据库时的用户名称和密码。代码如下。

```
string connectionString = "Data Source=SJB;Initial Catalog=master;User
ID=sa;Pwd = 123456";
```

然后通过 new 关键字创建 SqlConnection 对象的实例，将连接字符串作为参数进行传递。代码如下。

```
SqlConnection conn = new SqlConnection(connectionString);
```

【范例 2】

SqlConnection 对象可以直接创建而不传递连接字符串，创建后通过对象的属性设置连接字符串。代码如下。

```
SqlConnection conn = new SqlConnection();
connection.ConnectionString = connectionString;
```

9.2.2 SqlConnection 对象的属性

SqlConnection 对象包含多个属性，范例 2 中的 ConnectionString 对象用于获取或

设置 SQL Server 数据库的字符串。除了该属性外，如表 9-3 所示为该对象的其他常用属性。

表 9-3 SqlConnection 对象的常用属性

属性名称	说明
ConnectionTimeout	获取在尝试建立连接时终止尝试并生成错误之前所等待的时间
Database	获取当前数据库或连接打开后要使用的数据库的名称
DataSource	获取要连接的 SQL Server 实例的名称
WorkstationId	获取标识数据客户端的一个字符串
ServerVersion	获取包含客户端连接的 SQL Server 实例版本的字符串

【范例 3】

实例化 SqlConnection 对象，然后调用表 9-3 中的属性获取属性值，并将获取到的值输出。代码如下。

```
string connectionString = "Data Source=SJB;Initial Catalog=master;User
ID=sa;Pwd = 123456";
SqlConnection conn = new SqlConnection(connectionString);
conn.Open();
Response.Write("ConnectionString 属 性 值 : " + conn.ConnectionString
+"<br/>");
Response.Write("ConnectionTimeout 属性值: " + conn.ConnectionTimeout +
"<br/>");
Response.Write("Database 属性值: " + conn.Database + "<br/>");
Response.Write("ServerVersion 属性值: " + conn.ServerVersion);
```

9.2.3 SqlConnection 对象的方法

SqlConnection 对象包含多个方法，如范例 3 使用的 Open()方法，表 9-4 列举了常用的方法，并对这些方法进行说明。

表 9-4 SqlConnection 对象的常用方法

方法名称	说明
Open()	使用 ConnectionString 属性所指定的值打开数据库连接
Close()	关闭与数据库的连接，它是关闭任何打开连接的首选方法
Dispose()	释放当前所使用的资源
CreateCommand()	创建并返回一个与 SqlConnection 关联的 SqlCommand 对象

9.3 SqlCommand 对象

SqlCommand 对象表示执行操作数据库的命令，这个命令可以是 SQL 语句，也可以是存储过程。通过 SqlCommand 对象执行的操作类型有多个，如查询单条或多条数据、添加数据、删除数据、修改数据以及存储过程等。

9.3.1 创建 SqlCommand 对象

SqlConnection 对象提供 CreateCommand()方法，通过该方法可以创建并返回一个 SqlCommand 对象。代码如下。

```
    SqlCommand command = conn.CreateCommand();
```

除了调用上述方法外，还可以直接通过 new 创建 SqlCommand 对象。实例化 SqlCommand 对象时，有以下 4 种构造函数。

```
    SqlCommand command = new SqlCommand();
    SqlCommand command = new SqlCommand(string cmdText);
    SqlCommand command = new SqlCommand(string cmdText, SqlConnection conn);
    SqlCommand command = new SqlCommand(string cmdText, SqlConnection conn,
    SqlTransaction trans);
```

其中，cmdText 表示执行添加、查询、删除和搜索的 SQL 语句或者存储过程；conn 表示 SqlConnection 的实例对象；trans 表示 SqlTransaction 的实例对象。

【范例 4】

首先创建 SqlConnection 对象的实例，接着创建 SqlCommand 对象，在创建时传入两个参数，第一个参数表示执行的 SQL 语句，第二个参数表示 SqlConnection 对象。代码如下。

```
    string connectionString = "Data Source=SJB;Initial Catalog=master;User
ID=sa;Pwd = 123456";
    SqlConnection conn = new SqlConnection(connectionString);
    conn.Open();
    SqlCommand command = new SqlCommand("SELECT * FROM spt_monitor;", conn);
```

9.3.2 SqlCommand 对象的属性

创建 SqlCommand 对象之后，可以调用该对象的属性进行设置，常用的属性及其说明如表 9-5 所示。

表 9-5 SqlCommand 对象的常用属性

属性名称	说明
CommandText	获取或设置要对数据源执行的 Transact-SQL 语句或存储过程
CommandTimeout	获取或设置在终止执行命令的尝试并生成错误之前的等待时间
CommandType	获取或设置一个值，该值指示如何解释 CommandText 属性
Connection	获取或设置 SqlCommand 的此实例使用的 SqlConnection
Parameters	获取 SqlParameterCollection
Transaction	获取或设置将在其中执行 SqlCommand 的 SqlTransaction
UpdatedRowSource	获取或设置命令结果在由 DbDataAdapter 的 Update()方法使用时如何应用于 DataRow

在表 9-5 中，CommandType 属性的值是 System.Data.CommandType 枚举值之一，取值说明如下。

（1）Text：SQL 文本命令。

（2）TableDirect：表的名称。

（3）StoredProcedure：存储过程的名称。

【范例 5】

创建 SqlCommand 对象的实例，然后设置该对象的 Connection、CommandText 和 CommandType 等属性。代码如下。

```
SqlCommand command = new SqlCommand();
command.Connection = conn;
command.CommandText = "SELECT * FROM spt_monitor;";
command.CommandType = System.Data.CommandType.Text;
```

9.3.3 SqlCommand 对象的方法

SqlCommand 对象还提供了一系列的方法，通过这些方法可以完成对数据库的添加、删除、修改以及查询操作。如表 9-6 所示列出了 SqlCommand 对象的常用方法，并对这些方法进行说明。

表 9-6 SqlCommand 对象的常用方法

方法名称	说明
CreateParameter()	创建 SqlParameter 对象的新实例
Equals()	确定两个 Object 实例是否相等
ExecuteNonQuery()	对连接执行 Transact-SQL 语句并返回受影响的行数
ExecuteReader()	将 CommandText 发送到 Connection 并生成一个 SqlDataReader
ExecuteScalar()	执行查询，并返回查询所返回的结果集中第一行的第一列，忽略其他列或行
ExecuteXmlReader()	将 CommandText 发送到 Connection 并生成一个 XmlReader 对象
EndExecuteNonQuery()	完成 Transact-SQL 语句的异步执行
EndExecuteReader()	完成 Transact-SQL 语句的异步执行，返回请求的 SqlDataReader

【范例 6】

ExecuteScalar()方法执行查询，并返回查询所返回的结果集中第一行的第一列。如果有多行多列，则忽略其他列或行。简单来说，ExecuteScalar()方法返回第一行第一列的结果。下面通过该方法查询 master 数据库中 spt_monitor 表中的总记录，然后将结果输出。代码如下。

```
string connectionString = "Data Source=SJB;Initial Catalog=master;User
ID=sa;Pwd = 123456";
SqlConnection conn = new SqlConnection(connectionString);
                                               //创建 SqlConnection 对象
conn.Open();                                   //打开数据库连接
SqlCommand command = conn.CreateCommand();     //创建 SqlCommand 对象
command.CommandText = "SELECT COUNT(*) FROM spt_monitor;";
```

```
command.CommandType = System.Data.CommandType.Text;
int total = Convert.ToInt32(command.ExecuteScalar());
Response.Write("spt monitor 表中的总记录：" + total);
conn.Close();                                              //关闭数据库连接
```

在使用 SqlCommand 对象操作数据库表之前，必须确保已经成功连接数据库，使用 SqlCommand 对象的一般步骤如下。

（1）创建数据库连接对象。

（2）创建 SqlCommand 的实例对象。

（3）执行 SQL 语句或者存储过程。

（4）关闭数据库连接。

9.4 SqlParameter 对象

一般来说，在更新 DataTable 或者是 DataSet 时，如果不采用 SqlParameter 对象，那么当输入的 SQL 语句出现歧义时，如字符串中含有单引号，程序就会发生错误，而且其他人可以轻易通过拼接 SQL 语句来进行注入攻击。

SqlParameter 对象表示 SqlCommand 的参数，也可以是它到 DataSet 列的映射。SqlParameter 类不能被继承。

9.4.1 创建 SqlParameter 对象

创建 SqlParameter 对象有两种方法：一种是使用 SqlCommand 对象的 CreateParameter()方法；第二种是使用 new 关键字。通过 new 关键字创建 SqlParameter 对象时，SqlParameter 对象有 7 种构造函数，常用的两种形式如下。

```
SqlParameter para = new SqlParameter();
SqlParameter para = new SqlParameter(string parameterName, object value);
```

使用第二种构造形式时需要传入两个参数，parameterName 指定参数名，它是在 SQL 语句里要替换掉的名称；value 表示一个任意数据库支持的类型对象，具体的类型系统自动判断。

【范例 7】

创建 SqlParameter 对象，并在创建时指定参数名和参数值。代码如下。

```
SqlParameter para = new SqlParameter("@username", "张三");
```

9.4.2 SqlParameter 对象的属性

SqlParameter 对象包含一系列的属性和方法，如表 9-7 所示列举了该对象的常用属性。

表 9-7 SqlParameter 对象的常用属性

属性名称	说明
IsNullable	获取或设置一个值，该值指示参数是否接受空值
ParameterName	获取或设置 SqlParameter 的名称
SqlValue	获取作为 SQL 类型的参数的值，或设置该值
TypeName	获取或设置表值参数的类型名称
Value	获取或设置该参数的值

【范例 8】

定义 SqlParameter 类型的数组，然后遍历该数组。在 foreach 语句进行遍历时，通过 ParameterName 获取参数的名称，SqlValue 获取参数的值。代码如下。

```
SqlParameter[] parm = new SqlParameter[]{  //定义 SqlParameter 类型数组
    new SqlParameter("name1","张云"),       //为指定变量赋值
    new SqlParameter("name2","许飞"),
    new SqlParameter("name3","李遥遥"),
};
foreach (SqlParameter item in parm) {
    Response.Write("获取到的 SqlParameter 对象的名称:" + item.ParameterName+
    "<br/>");
    Response.Write("获取到作为 SQL 类型参数的值: " + item.SqlValue+"<br/>");
    Response.Write("==========================================<br/>");
}
```

9.5 实验指导——在数据库表中添加记录

添加、修改、删除和查询是对数据库的主要操作，本节实验指导利用前面的对象实现在数据库表中添加数据记录。步骤如下。

（1）创建 AddData.aspx 窗体，在窗体的表单元素中创建七行两列的表格。

（2）在第一行的表格中添加内容，第一列表示“用户名”文本，第二列表示【用户名】输入框。代码如下。

```
<td align="right">用户名: </td>
<td><asp:TextBox ID="txtUserName" runat="server" CssClass="txt"></asp:
TextBox></td>
```

（3）在第二行的表格中添加内容，第一列表示“密码”文本，第二列表示【密码】输入框。代码如下。

```
<td align="right">密码: </td>
<td><asp:TextBox ID="txtPass" TextMode="Password" runat="server" CssClass
="txt"></asp:TextBox></td>
```

（4）在第三行的表格中添加内容，第一列表示“电子邮箱”文本，第二列表示【电子邮箱】输入框，其 TextMode 属性值为 Email。代码如下。

```
<td align="right">电子邮箱：</td>
<td><asp:TextBox ID="txtEmail" TextMode="Email" runat="server" CssClass=
"txt"></asp:TextBox></td>
```

（5）在第四行的表格中添加内容，第一列表示“出生日期”文本，第二列表示【出生日期】输入框，其 TextMode 属性值为 Date。代码如下。

```
<td align="right">出生日期：</td>
<td><asp:TextBox ID="txtBirth" TextMode="Date" runat="server" CssClass=
"txt"></asp:TextBox></td>
```

（6）在第五行的表格中添加内容，第一列表示“个人说明”文本，第二列表示【个人说明】输入框。代码如下。

```
<td align="right">个人说明：</td>
<td><asp:TextBox ID="txtIntro" runat="server" Width="250" Height="50">
</asp:TextBox></td>
```

（7）在第六行的表格中添加 Button 控件，代码如下。

```
<asp:Button ID="btnRegister" runat="server" Text=" 注 册 " OnClick=
"btnRegister_Click" />
```

（8）在第七行的表格中添加 Label 控件，代码如下。

```
<asp:Label ID="lblResult" runat="server" ForeColor="Blue" Font-Size
="Large"></asp:Label>
```

（9）在 AddData.aspx.cs 后台页面添加 Button 控件的 Click 事件代码，首先创建 SqlConnection 对象的实例，代码如下。

```
protected void btnRegister Click(object sender, EventArgs e) {
    string connectionString = "Data Source=SJB;Initial Catalog=User
    Register;User ID=sa;Pwd = 123456";
    SqlConnection conn = new SqlConnection(connectionString);
    /* 省略其他代码 */
}
```

（10）调用 SqlConnection 对象的 Open()方法打开数据库连接，这里省略代码。

（11）声明执行添加操作的 SQL 语句，并且通过 SqlParameter 对象创建参数，这些参数的值是从用户界面接收的。代码如下。

```
string sql = "INSERT INTO UserInfo(UserName,UserPass,UserMail,UserBirth,
UserIntro) VALUES(@name,@pass,@mail,@birth,@intro)";
SqlParameter[] parm = new SqlParameter[]{
    new SqlParameter("@name",txtUserName.Text),     //为指定变量赋值
    new SqlParameter("@pass",txtPass.Text),
    new SqlParameter("@mail",txtEmail.Text),
    new SqlParameter("@birth",txtBirth.Text),
    new SqlParameter("@intro",txtIntro.Text)
};
```

（12）创建 SqlCommand 对象，指定该对象的 CommandText、CommandType 和 Connection 属性，并将 SqlParameter 对象中的参数添加到 Parameters 中。代码如下。

```
SqlCommand command = new SqlCommand();
command.CommandText = sql;
command.CommandType = System.Data.CommandType.Text;
command.Connection = conn;
foreach (SqlParameter item in parm) {
    command.Parameters.Add(item);
}
```

（13）调用 SqlCommand 对象的 ExecuteNonQuery()方法执行添加操作，如果添加成功显示提示，并将输入框的值清空。代码如下。

```
try {
    int result = command.ExecuteNonQuery();
    if (result > 0) {
        lblResult.Text = "添加成功，请到数据库查看";
        txtUserName.Text = "";
        //省略其他代码
    } else {
        lblResult.Text = "添加失败";
    }
} catch (Exception ex) {
    lblResult.Text = "添加失败，失败原因是：" + ex.Message;
}
```


（14）调用 SqlConnection 对象的 Close()方法关闭数据库连接，这里省略代码。

（15）运行 AddData.aspx 页面查看效果，如图 9-1 所示。



图 9-1　页面运行效果

（16）在图 9-1 中输入内容完毕后单击【注册】按钮，如图 9-2 所示。



图 9-2 注册成功时的效果

9.6 SqlDataReader 对象

ADO.NET 中提供了 SqlDataReader 对象，该对象用于从数据库中读取数据。在使用 SqlDataReader 对象读取数据时，必须要与数据库保持连接，断开连接后不能再读取数据。

9.6.1 创建 SqlDataReader 对象

SqlDataReader 对象读取数据库表中的数据时，数据连接必须处于打开状态，而且每次从查询结果中读取一行数据到内存中。使用 SqlDataReader 对象时，需要注意以下几点。

（1）SqlDataReader 只能读取数据，不能对数据库执行任何修改或者插入操作。

（2）SqlDataReader 只能向前读取数据，不能回头读取已经被访问的数据。

（3）SqlDataReader 对象直接把数据传递到其他对象或窗体页面。

直接通过 SqlCommand 对象的 ExecuteReader()方法可以创建 SqlDataReader 对象，代码如下。

```
SqlDataReader dr = command.ExecuteReader();
```

9.6.2 SqlDataReader 对象的属性

调用 SqlDataReader 对象的属性可以获取当前行中的列数、当前行的嵌套深度，如表 9-8 所示的对 SqlDataReader 对象的常用属性的说明。

表 9-8 SqlDataReader 对象的常用属性

属性名称	说明
Depth	获取一个值，用于指示当前行的嵌套深度
FieldCount	获取当前行中的列数
HasRows	获取一个值，该值指示 SqlDataReader 中是否包含一行或多行

续表

属性名称	说明
IsClosed	检索一个布尔值，该值指示是否已关闭指定的 SqlDataReader 实例
VisibleFieldCount	获取 SqlDataReader 中未隐藏的字段的数目

【范例 9】

HasRows 属性返回一个布尔值，使用代码如下。

```
if (dr.HasRows) {
    //如果包含行
}
```

9.6.3 SqlDataReader 对象的方法

SqlDataReader 对象还提供了一系列的方法，常用方法的说明如下。

（1）Close()方法：关闭 SqlDataReader 对象，释放资源。

（2）Read()方法：返回一个布尔值，使用 SqlDataReader 指针前进到下一条记录，如果前进成功则返回 true，否则返回 false。

（3）isDBNull()方法：获取一个值，该值指示列中是否包含不存在的或缺少的值。

【范例 10】

通过 Read()方法读取数据，在读取数据的过程中，并不能保证读取的字段中都有数据，因此，可以通过 DBNull 对象的 Value 属性判断。代码如下。

```
if(read["st brithday"] == DBNull.Value) {
    //数据为空时的代码
}
```

9.7 实验指导——读取数据库表中的记录

9.6 节简单介绍了 SqlDataReader 对象，通过该对象可以读取数据库中的数据。使用 SqlDataReader 对象读取数据的一般步骤如下。

（1）创建数据库连接。

（2）创建执行时的 SQL 语句或者存储过程。

（3）创建 SqlCommand 的实例对象。

（4）打开数据库连接并创建 SqlDataReader 的实例对象。

（5）使用 Read()方法逐行读取数据。

（6）关闭 SqlDataReader 的实例对象。

（7）关闭数据库连接。

本节通过 SqlDataReader 对象读取 UserRegister 数据库中 UserInfo 表的全部记录，并将这些记录显示到页面。步骤如下。

（1）创建 ReadData.aspx 页面，在页面的表单元素中添加表格元素。其中，thead 显示标题，tbody 显示读取的内容。代码如下。

```
<table width="80%" height="200" border="1">
    <thead>
        <tr>
            <td>用户名</td>
            <td>用户密码</td>
            <td>电子邮箱</td>
            <td>出生日期</td>
            <td>个人说明</td>
        </tr>
    </thead>
    <tbody id="tbody" runat="server"></tbody>
</table>
```

（2）在 ReadData.aspx.cs 后台页面添加代码，首先创建 SqlConnection 对象的实例并打开连接，接着创建 SqlCommand 对象的实例，然后调用 ExecuteReader()方法创建 SqlDataReader 对象，创建完毕后通过 Read()方法读取数据，最后将读取的结果显示到页面的 tbody 元素中。代码如下。

```
string connectionString = "Data Source=SJB;Initial Catalog=UserRegister;
User ID=sa;Pwd = 123456";
using (SqlConnection conn = new SqlConnection(connectionString)) {
    conn.Open();
    SqlCommand command = new SqlCommand("SELECT * FROM UserInfo", conn);
    string message = "";
    using (SqlDataReader dr = command.ExecuteReader()) {
        while (dr.Read()) {
            if (dr["UserIntro"] == DBNull.Value || string.IsNullOrEmpty
            (dr["UserIntro"].ToString())) {
                message += "<tr><td>" + dr["UserName"].ToString() + "</td>
                <td>" + dr["UserPass"].ToString() + "</td><td>" + dr["User
                Mail"].ToString() + "</td><td>" + Convert.ToDateTime(dr
                ["UserBirth"]).ToString("yyyy年MM月dd日") + "</td><td>暂
                无</td></tr>";
            } else {
                message += "<tr><td>" + dr["UserName"].ToString() + "</td>
                <td>" + dr["UserPass"].ToString() + "</td><td>" + dr["User
                Mail"].ToString() + "</td><td>" + Convert.ToDateTime(dr
                ["UserBirth"]).ToString("yyyy年MM月dd日") + "</td><td>" +
                dr["UserIntro"].ToString() + "</td></tr>";
            }
        }
    }
    tbody.InnerHtml = message;
}
```

（3）运行 ReadData.aspx 页面查看效果，如图 9-3 所示。



图 9-3 读取数据库表中的记录

9.8 DataSet 对象

SqlDataReader 对象可以读取数据库表中的记录，但是它必须与数据库创建连接，数据库断开连接时不能使用 SqlDataReader 对象。ADO.NET 提供了 DataSet 对象，该对象在断开数据库连接时也可以进行读取操作。

9.8.1 DataSet 工作原理

DataSet 对象表示数据集，它可以保存数据库中读取的数据，在数据保存之后可以被直接操作。DataSet 在 ADO.NET 断开连接的分布式数据中起到了重要作用，它是数据驻留在内存中的表示形式，不管数据源是什么，它都可以提供一致的关系编程模型。如图 9-4 所示为 DataSet 对象的工作原理。

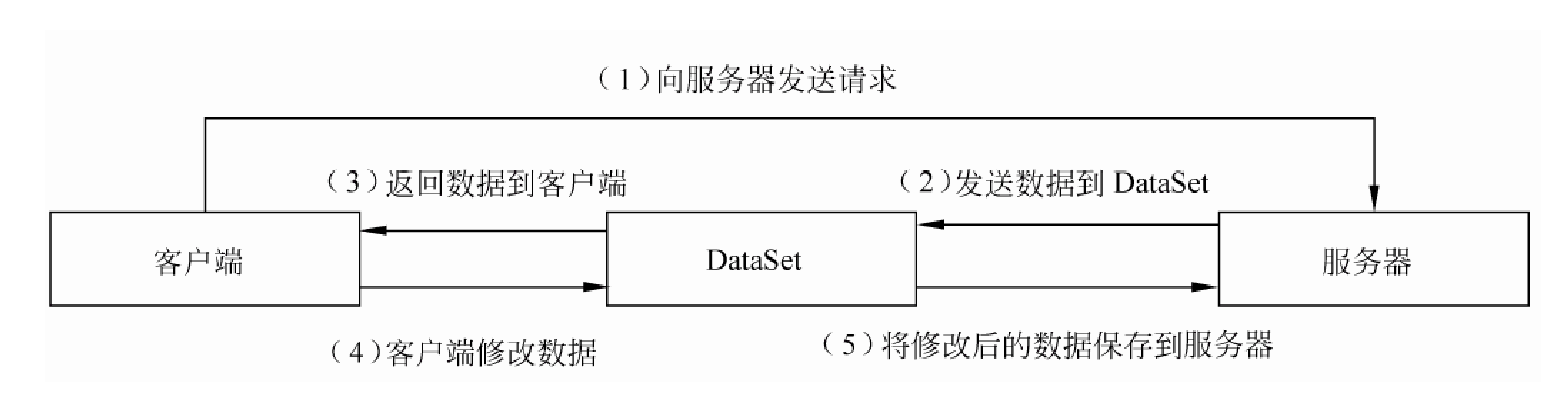


图 9-4 DataSet 对象的工作原理

在图 9-4 中，当应用程序需要数据时会向服务器发出请求获取数据，服务器先将数据发送到 DataSet 中，然后再将数据集传递给客户端，客户端将数据集中的数据修改后，会统一将修改过的数据集发送到服务器，服务器接收并修改数据库中的数据。

9.8.2 创建 DataSet 对象

直接通过 new 关键字实例化 DataSet 对象时，可以通过以下两种形式进行创建。

```
DataSet ds = new DataSet();                    //直接创建
DataSet ds = new DataSet("Customer");          //通过数据集名称创建
```

其中，第一种构造函数经常用到。在第二种构造函数中，在创建 DataSet 对象时需要传入数据集的名称。

9.8.3 DataSet 对象的属性

DataSet 对象包含一系列的属性和方法，只有将数据填充到该对象后，这个对象才能使用。如表 9-9 所示列举了 DataSet 对象的常用属性，并对这些属性进行说明。

表 9-9 DataSet 对象的常用属性

属性名称	说明
DataSetName	获取或设置当前 DataSet 的名称
DefaultViewManager	获取 DataSet 所包含的数据的自定义视图，以允许使用自定义的 DataViewManager 进行筛选、搜索和导航
ExtendedProperties	获取与 DataSet 相关的自定义用户信息的集合
Relations	获取用于将表链接起来并允许从父表浏览到子表的关系的集合
Tables	获取包含在 DataSet 中的表的集合

在表 9-9 中，Tables 属性最经常被用到，通过该属性获取包含在 DataSet 中的表的集合。其中，ds.Tables[0]表示获取第一个表的数据，ds.Tables[1]表示获取第二个表的数据，其返回结果是一个 DataTable 对象。

9.8.4 DataSet 填充数据

创建 DataSet 对象完毕后，可以调用相应的方法填充数据。DataSet 填充或者修改数据时经常使用 SqlDataAdapter 对象。通过 SqlDataAdapter 对象填充 DataSet 对象的一般步骤如下。

（1）创建数据库连接对象。

（2）创建要执行的 SQL 语句或者存储过程。

（3）利用 SQL 语句和数据库连接对象创建 SqlDataAdapter 对象。

（4）向 DataSet 对象中填充数据。

【范例 11】

更改 9.7 节中实验指导的内容，通过 DataSet 对象获取 UserInfo 表中的全部记录，并将获取到的记录显示到页面。步骤如下。

（1）创建 SetData.aspx 页面，在页面的表单元素中添加表格元素。其中，thead 显示

标题，tbody 显示读取的内容，具体代码不再显示。

（2）在 SetData.aspx.cs 后台页面的 Load 事件中添加代码，通过 SqlDataAdapter 对象向 DataSet 填充数据。代码如下。

```
string connectionString = "Data Source=SJB;Initial Catalog=User
Register;User ID=sa;Pwd = 123456";
SqlConnection connection = new SqlConnection(connectionString);
                                        //创建 SqlCommand 对象
string sql = "SELECT UserName,UserPass,UserMail,UserBirth,UserIntro FROM
UserInfo";                              //声明 SQL
SqlDataAdapter da = new SqlDataAdapter(sql, connection);
                                        //SQL 语句和 connection 创建对象
DataSet ds = new DataSet();
da.Fill(ds);
```

（3）继续添加代码，遍历 DataSet 中的数据将其输出显示到页面。遍历数据时需要通过两个 foreach 语句，外层 foreach 语句遍历行，内层 foreach 语句遍历列。通过 ds.Tables[0].Rows 获取表中行的集合，通过 ds.Tables[0].Columns 获取表中列的集合。代码如下。

```
string result = "";
foreach (DataRow mDr in ds.Tables[0].Rows) {                //遍历行
   result += "<tr>";
   foreach (DataColumn mDc in ds.Tables[0].Columns) {      //遍历列
      result += "<td>";
      if (string.IsNullOrEmpty(mDr[mDc].ToString())) {
         result += "暂无";
      } else {
         result += mDr[mDc].ToString();
      }
      result += "</td>";
   }
   result += "</tr>";
}
tbody.InnerHtml = result;                              //输出数据添加到页面
```

（4）运行页面查看效果，具体效果图不再显示。

9.8.5 DataSet 与 SqlDataReader 的区别

DataSet 对象和 SqlDataReader 对象都可以保存数据，它们功能相似，但是并不能相互替换。下面分别从数据库连接、处理数据的速度和内存占用等方面进行介绍。

（1）数据库连接：DataSet 对象可以不和数据库创建连接，把表全部读到 SQL 中的缓冲池，并断开和数据库的连接；而 SqlDataReader 对象必须与数据库进行连接，读取表时，只能向前读取，读取完毕后由用户决定是否断开连接。

（2）处理数据的速度：DataSet 对象读取和处理数据的速度较慢，而 SqlDataReader

读取和处理数据的速度较快。

（3）更新数据库：DataSet 对象对数据集中的数据更新后，可以把数据库中的数据更新；而 SqlDataReader 只能读取数据，不能对数据库中的数据更新。

（4）内存占用：DataSet 对象占用内存较多；而 SqlDataReader 对象占用内存较少。

（5）是否支持分页和排序功能：DataSet 支持分页和排序；而 SqlDataReader 不支持。

9.9 SqlDataAdapter 对象

在前面已经提到，向 DataSet 对象中填充数据时需要使用 DataAdapter 对象。DataAdapter 对象是一个适配器，它表示一组操作数据的命令和一个数据库连接。该对象充当数据库和 DataSet 之间的桥梁，用以协调双方数据同步。

9.9.1 创建 SqlDataAdapter 对象

与创建其他对象的方法一样，可以通过 new 实例化 SqlDataAdapter 对象。它有 4 种构造函数，形式如下。

```
SqlDataAdapter da = new SqlDataAdapter();
SqlDataAdapter da = new SqlDataAdapter(SqlCommand selectCommandText);
SqlDataAdapter  da  =  new  SqlDataAdapter(String  selectCommandText,
SqlConnection selectCommand);
SqlDataAdapter da = new SqlDataAdapter(String selectCommandText, String
selectConnectionString);
```

其中，第一行表示初始化 SqlDataAdapter 类的实例；第二行表示创建该类的实例，用指定的 SqlCommand 作为 SelectCommand 的属性；第三行使用 SelectCommand 和 SqlConnection 对象初始化一个 SqlDataAdapter 类的实例；最后一行用 SelectCommand 和一个连接字符串初始化 SqlDataAdapter 类的实例。

9.9.2 SqlDataAdapter 对象更新数据

SqlDataAdapter 对象提供多个属性，创建对象完毕后可以使用这些属性。如表 9-10 所示列举了 SqlDataAdapter 对象的常用属性。

表 9-10 SqlDataAdapter 对象的常用属性

属性名称	说明
DeleteCommand	获取或设置一个 SQL 语句或存储过程，以从数据集删除记录
InsertCommand	获取或设置一个 SQL 语句或存储过程，以在数据源中插入新记录
SelectCommand	获取或设置一个 SQL 语句或存储过程，用于在数据源中选择记录
UpdateCommand	获取或设置一个 SQL 语句或存储过程，以更新数据源中的记录
TableMappings	获取一个集合，它提供源表和 DataTable 之间的主映射

> **提 示**
>
> 除表 9-10 中列出的属性外，SqlDataAdapter 还提供方法，其中 Update()方法和 Fill()方法最为常用。Update()方法为 DataSet 中每个已插入、已更新或者已删除的行调用相应的 INSERT、UPDATE 或者 DELETE 语句；Fille()方法填充数据到 DataSet 或者 DataTable 中。

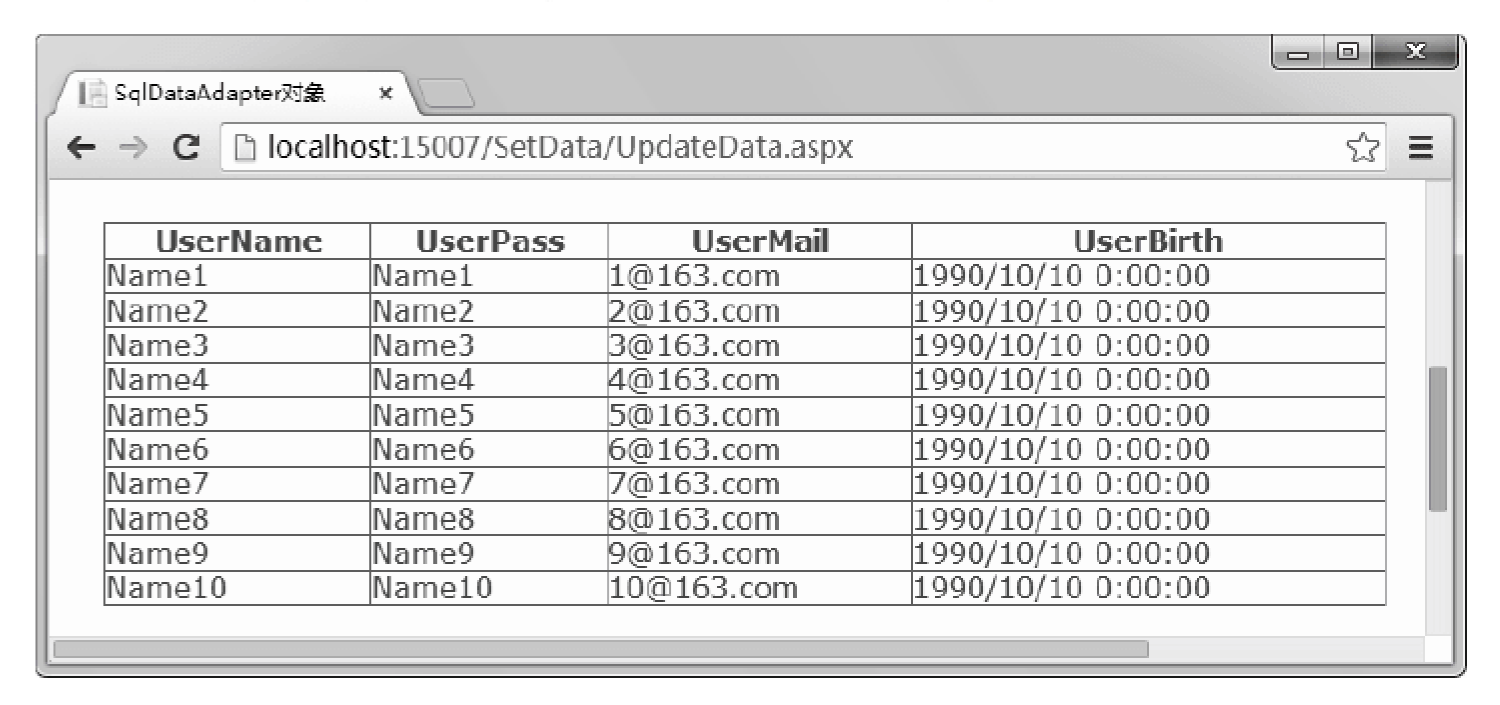
【范例 12】

开发者可以通过 SqlDataAdapter 对象的 Update()方法实现更新，通过这种方式实现更新有三种情况：即 SqlCommandBuilder 自动生成更新；使用配置数据源方式更新；手动编写命令。下面通过 SqlCommandBuilder 实现更新的代码，并将更新后的结果显示到 GridView 控件中。代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    string constr = "Data Source=SJB;Initial Catalog= UserRegister;User
    ID=sa;Pwd = 123456";
    SqlConnection conn = new SqlConnection(constr);
    SqlCommand selectCMD = new SqlCommand("SELECT TOP 0 UserName, UserPass,
    UserMail,UserBirth FROM UserInfo", conn);
    DataTable dt = new DataTable();
    SqlDataAdapter sda = new SqlDataAdapter(selectCMD);
    sda.Fill(dt);
    for (int i = 1; i <= 10; i++)            //给 DataTable 添加 10 条记录
        dt.Rows.Add(new object[] { "Name" + i, "Name" + i, i + "@163.com",
        "1990-10-10" });
    SqlCommandBuilder scb = new SqlCommandBuilder(sda);
    sda.Update(dt.GetChanges());             //执行更新
    dt.AcceptChanges();                      //使 DataTable 保存更新
    gvShow.DataSource = dt;
    gvShow.DataBind();
}
```

在上述代码中，创建 SqlCommand 对象时设置 SELECT 查询命令，SqlCommand Builder 要求至少有 SELECT 命令。而 SELECT TOP 0 不是为了查询出数据，是要查询出表结构以向 DataTable 中填充表结构。

运行上述代码所在的页面查看效果，如图 9-5 所示。



UserName	UserPass	UserMail	UserBirth
Name1	Name1	1@163.com	1990/10/10 0:00:00
Name2	Name2	2@163.com	1990/10/10 0:00:00
Name3	Name3	3@163.com	1990/10/10 0:00:00
Name4	Name4	4@163.com	1990/10/10 0:00:00
Name5	Name5	5@163.com	1990/10/10 0:00:00
Name6	Name6	6@163.com	1990/10/10 0:00:00
Name7	Name7	7@163.com	1990/10/10 0:00:00
Name8	Name8	8@163.com	1990/10/10 0:00:00
Name9	Name9	9@163.com	1990/10/10 0:00:00
Name10	Name10	10@163.com	1990/10/10 0:00:00

图 9-5 SqlDataAdapter 对象更新数据

9.10 其他常用对象

除了前面介绍的对象外，ADO.NET 还提供了其他的对象，下面介绍两种常用的对象：DataTable 对象和 DataView 对象。

9.10.1 DataTable 对象

DataTable 对象与 DataSet 对象一样，将数据库中的数据提取出来进行保存，以提供断开连接时对数据的访问。但是它们之间也有区别：DataTable 对象保存单个表；DataSet 对象可以保存多个表。在数据显示时，使用 DataTable 能够更加简单地控制数据显示。

开发者可以通过两种方式创建 DataTable 对象，第一种是通过 DataSet 对象的 Tables 属性获取，代码如下。

```
DataTable dt = ds.Tables[0];
```

第二种是通过动态创建的方式进行创建，这需要利用 DataSet 对象的相关属性和方法。动态创建 DataTable 对象的一般步骤如下。

（1）创建 DataTable 的实例对象。

（2）创建 DataColumn 对象构建表结构。

（3）将创建好的表结构添加到 DataTable 对象中。

（4）调用 DataTable 对象的 NewRow()方法创建 DataRow 对象。

（5）在 DataRow 对象中添加一条或者多条数据记录。

（6）将数据插入到 DataTable 对象中。

【范例 13】

通过下面的代码动态创建一个五行两列的表格，并将表格中的内容输出到页面。步骤如下。

（1）通过 new 创建 DataTable 的实例对象，代码如下。

```
DataTable dt = new DataTable();
```

（2）创建两个 DataColumn 对象，指定这两个对象的 ColumnName、Unique、Caption 和 ReadOnly 属性。以第一个对象为例，代码如下。

```
DataColumn column1 = new DataColumn("name", typeof(string));
column1.ColumnName = "name";
column1.Unique = true;
column1.Caption = "ID";
column1.ReadOnly = true;
```

（3）调用 dt.Columns.Add()方法将创建好的表结构添加到 DataTable 对象，代码如下。

```
dt.Columns.Add(column1);
```

（4）在 for 语句中调用 NewRow()方法创建 DataRow 对象，并为 DataRow 对象添加

记录，然后调用 dt.Rows.Add()方法将数据插入到 DataTable 对象中，代码如下。

```
DataRow row;
for (int i = 1; i <= 5; i++) {
    row = dt.NewRow();                    //创建 DataRow 对象
    row["name"] = "老师" + i;
    row["age"] = "年龄" + i;
    dt.Rows.Add(row);
}
```

（5）创建 DataSet 的实例对象，并将 DataTable 添加到 DataSet 中，代码如下。

```
DataSet ds = new DataSet();
ds.Tables.Add(dt);
```

（6）通过 foreach 语句循环 DataSet 对象中的数据，并将数据显示到页面，代码如下。

```
string result = "<table border=\"1\">";
foreach (DataRow mDr in ds.Tables[0].Rows) {                 //遍历行
    result += "<tr>";
    foreach (DataColumn mDc in ds.Tables[0].Columns) {       //遍历列
        result += "<td>"+mDr[mDc].ToString()+"</td>";
    }
    result += "</tr>";
}
result += "</table>";
lblMessage.Text = result;
```


（7）运行页面查看效果，效果图不再显示。

DataTable 对象包含多个属性和方法，如表 9-11 所示列举了常用属性。

表 9-11 DataTable 对象的常用属性

属性名称	说明
Columns	获取属于表的所有列的集合
Rows	获取属于表的所有行的集合
DefaultView	获取或能包括筛选视图或游标位置的表的自定义视图
HasError	获取一个值，该值指示表所属的 DataSet 的任何表的任何行中是否有错误
MinimumCapacity	获取或设置表最初的起始大小
TableName	获取或设置 DataTable 的名称

9.10.2 DataView 对象

DataView 对象可用于排序、筛选、搜索、编辑和导航的 DataTable 的可绑定数据的自定义列，通常把 DataView 称为数据视图。一个 DataSet 中可以有多个 DataTable，而每一个 DataTable 对象都存在一个默认的数据视图。

开发者可以使用 DataTable 对象的 DefaultView 属性创建一个 DataView 对象。另外，开发者也可以自定义 DataView 表示 DataTable 中的数据的子集。

DataView 对象中包含多个属性和方法，如表 9-12 和表 9-13 所示分别对这些属性和方法进行了说明。

表 9-12 DataView 对象的常用属性

属性名称	说明
AllowDelete	用于获取或设置一个值，该值指示是否允许删除
AllowEdit	用于获取或设置一个值，该值指示是否允许编辑
AllowNew	用于获取或设置一个值，该值指示是否可以使用 AddNew()方法添加新行
ApplyDefaultSort	获取或设置一个值，该值指示是否使用默认排序
Count	用于 RowFilter 和 RowStateFilter 之后，获取 DataView 中记录的数量
RowFilter	获取或设置用于筛选在 DataView 中查看哪些行的表达式
RowStateFilter	获取或设置用于 DataView 中的行状态筛选器
Sort	用于获取或设置 DataView 中的一个或多个排序列以及排序顺序
Table	用户获取或设置源 DataTable

表 9-13 DataView 对象的常用方法

方法名称	说明
AddNew()	将新行添加到 DataView 对象中
Delete()	删除指定索引位置的行
FindRows()	返回 DataRowView 对象的数组，这些对象的列与指定的排序关键字值匹配
Close()	关闭 DataView 对象

【范例 14】

通过 SqlDataAdapter 对象填充数据，然后通过 DataView 对象筛选数据，并将筛选后的结果输出到页面。步骤如下。

（1）通过 SqlDataAdapter 对象向 DataSet 对象中填充数据，代码如下。

```
string connectionString = "Data Source=SJB;Initial Catalog=UserRegister;
User ID=sa;Pwd = 123456";
SqlConnection connection = new SqlConnection(connectionString);
//创建 SqlCommand 对象
string sql = "SELECT * FROM UserInfo";              //声明 SQL 语句
SqlDataAdapter da = new SqlDataAdapter(sql, connection);
                                        //SQL 语句和 connection 创建对象
DataSet ds = new DataSet();
da.Fill(ds);
```

（2）获取 DataSet 对象中的表，调用 DataTable 对象的 DefaultView 属性获取 DataView 对象，代码如下。

```
DataTable dt = ds.Tables[0];
System.Data.DataView dv = dt.DefaultView;
```

（3）设置 DataView 对象的 RowFilter 属性和 Sort 属性，代码如下。

```
dv.RowFilter = "UserId>=2 AND UserID<=4";
dv.Sort = "UserId desc";
```

（4）向窗体页面添加 GridView 控件，在后台中指定该控件的 DataSource 属性，并通过 DataBind()方法进行绑定，代码如下。

```
gvShow.DataSource = dt;
gvShow.DataBind();
```

（5）运行页面查看效果，如图 9-6 所示。从图 9-6 中可以看出，通过 DataView 对象筛选已经起到作用，已经查询出 UserId 字段的值大于等于 2 小于等于 4 的所有数据记录。

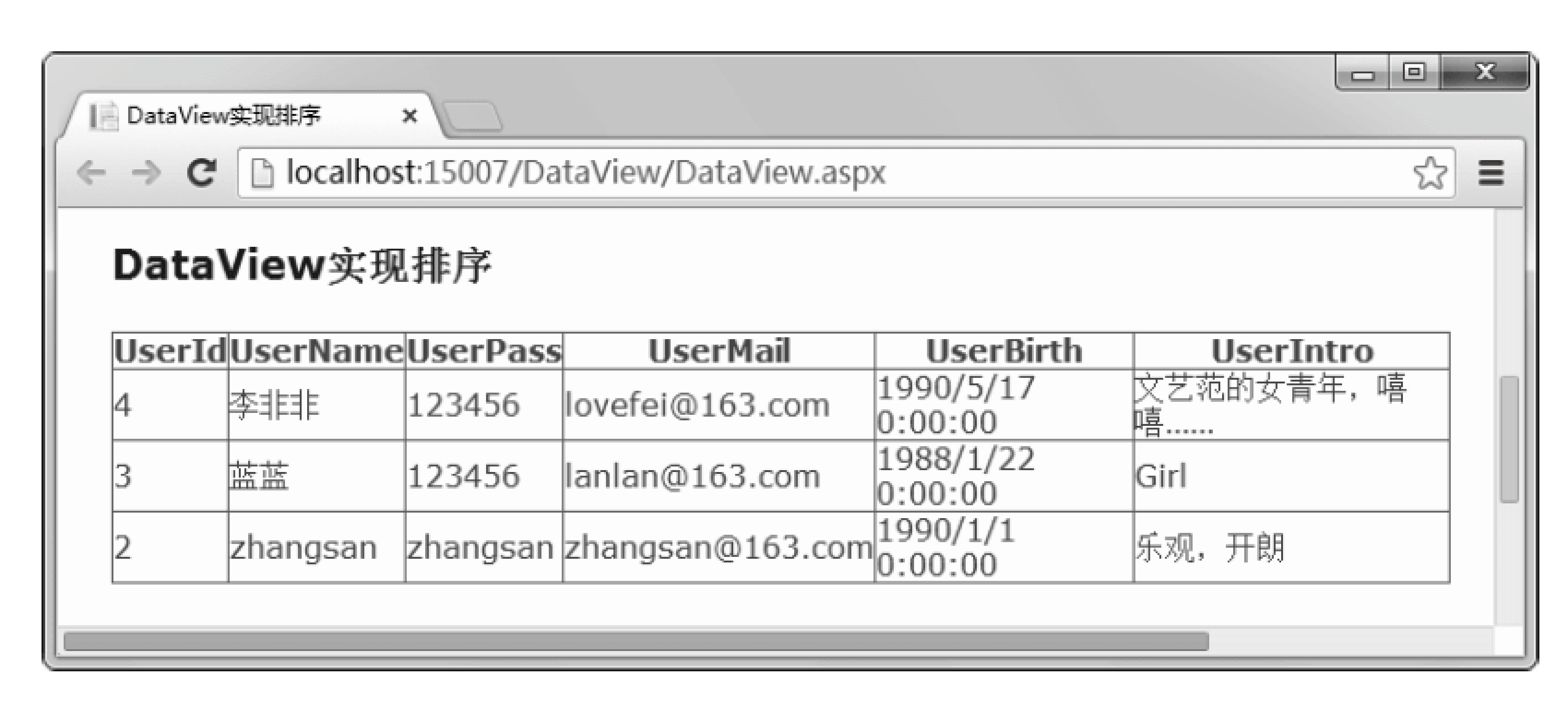


图 9-6 DataView 对象筛选数据

9.11 实验指导——创建公用的帮助类

开发者可以试想这样一个问题：在一个完整的系统或者网站中，需要多次对数据库表中的数据进行操作。每次操作数据时都需要创建数据库连接，这样，多次实现数据访问时会出现许多重复的代码。例如，每从都需要通过 SqlConnection 创建连接，执行添加、删除或修改操作时都需要调用 ExecuteNonQuery()方法。

针对上述问题，微软提供了一个 SqlHelper 帮助类，它是一个免费的、开源的类，开发者可以在使用时进行下载。可以将 SqlHelper 类理解为一个帮助类，该类包含对数据库的操作方法。如果开发者不想使用微软提供的 SqlHelper 类，可以自定义帮助类，在该类中将一些比较通用的方法封装起来，下次使用时可以直接调用类中的方法。

SqlHelper 类中的代码可以很简单，也可以很复杂。下面演示 SqlHelper 类的创建和使用，步骤如下。

（1）创建名称为 SqlHelper 的密封类，并在该类中声明一个 string 类型的静态只读 connString 变量，代码如下。

```
public sealed class SqlHelper {
    public static readonly string connString = " Data Source=SJB;Initial
    Catalog= UserRegister;User ID=sa;Pwd = 123456";
}
```

（2）在 SqlHelper 类中添加 ExecuteNonQuery()方法，该方法执行添加、删除和修改

数据，并且支持存储过程，代码如下。

```
public static int ExecuteNonQuery(string connString, CommandType commandType, string sql, params SqlParameter[] para) {
    using (SqlConnection conn = new SqlConnection(connString)) {
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = conn;
        cmd.CommandType = commandType;
        cmd.CommandText = sql;
        if (para != null) {
            foreach (SqlParameter sp in para) {
                cmd.Parameters.Add(sp);
            }
        }
        conn.Open();
        return cmd.ExecuteNonQuery();
    }
}
```

在上述代码中，ExecuteNonQuery()方法返回受影响的行数。使用该方法时需要传入4个参数：connString 表示连接字符串，可以通过 SqlHelper.connString 赋值；commandType 表示命令类型，如果是 SQL 语句，则为 CommandType.Text，如果是存储过程，则为 CommandType.StoredProcdure；sql 表示 SQL 语句或者存储过程的名称；para 表示 SQL 参数，如果没有参数，则为 null。

（3）在 SqlHelper 类中创建 ExecuteReader()方法，该方法是执行查询的方法，支持存储过程。代码如下。

```
public static SqlDataReader ExecuteReader(string connString, CommandType commandType, string sql, params SqlParameter[] para) {
    SqlConnection conn = new SqlConnection(connString);
    SqlDataReader dr = null;
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = conn;
    cmd.CommandType = commandType;
    cmd.CommandText = sql;
    if (para != null) {
        foreach (SqlParameter sp in para) {
            cmd.Parameters.Add(sp);
        }
    }
    try {
        conn.Open();
        dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
    } catch (Exception ex) {
        Console.WriteLine(ex.Message);
        conn.Close();
    }
```

```
        return dr;
    }
```

在上述代码中，ExecuteReader()方法返回一个读取器对象，即 SqlDataReader 对象。使用该方法时也需要传入 4 个参数，参数说明可以参考步骤（2）。

（4）在 SqlHelper 中创建 GetDataSet()方法，该方法也是一个执行查询的方法，支持存储过程，代码如下。

```
    public static DataSet GetDataSet(string connString, CommandType
    commandType, string sql, params SqlParameter[] para) {
        using (SqlConnection conn = new SqlConnection(connString)) {
            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = new SqlCommand();
            da.SelectCommand.Connection = conn;
            da.SelectCommand.CommandText = sql;
            da.SelectCommand.CommandType = commandType;
            if (para != null) {
                foreach (SqlParameter sp in para) {
                    da.SelectCommand.Parameters.Add(sp);
                }
            }
            DataSet ds = new DataSet();
            conn.Open();
            da.Fill(ds);
            return ds;
        }
    }
```


在上述代码中，GetDataSet()方法返回一个数据集对象。在使用这个方法时，需要传入 4 个参数，它们的说明可以参考步骤（2）。

（5）在 SqlHelper 类中创建 ExecuteScalar()方法，该方法执行查询单个值的方法，支持存储过程。代码如下。

```
    public static object ExecuteScalar(string connString, CommandType
    commandType, string sql, params SqlParameter[] para) {
        using (SqlConnection conn = new SqlConnection(connString)) {
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = conn;
            cmd.CommandType = commandType;
            cmd.CommandText = sql;
            if (para != null) {
                foreach (SqlParameter sp in para) {
                    cmd.Parameters.Add(sp);
                }
            }
            conn.Open();
            return cmd.ExecuteScalar();
        }
```

```
    }
```

（6）创建测试页面，在页面的后台直接调用 SqlHelper 类中的方法。代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    object obj = SqlHelper.ExecuteScalar(SqlHelper.connString, Command
    Type.Text, "SELECT COUNT(*) FROM UserInfo", null);
    Response.Write("返回第一行第一列的值：" + obj.ToString());
}
```

（7）运行窗体页面查看效果，效果图不再显示。

思考与练习

一、填空题

1．.NET Framework 数据提供程序和__________是 ADO.NET 包含的两个组件。

2．通过 SqlConnection 对象的__________方法可以创建 SqlCommand 对象。

3．SqlCommand 对象的__________方法对连接执行 Transact-SQL 语句并返回受影响的行数。

4．SqlParameter 对象的__________属性用于获取或设置参数名称。

5．在下面的代码中，横线处的内容是__________。

```
using (SqlConnection conn = new
SqlConnection(connString)){
    SqlCommand command = new
    SqlCommand(sql, conn);
    conn.Open();
    SqlDataReader reader = command.
    ExecuteReader();
    while (            ) {
       /* 省略读取的代码 */
    }
    reader.Close();
}
```

6．SqlDataAdapter 对象向 DataSet 中填充数据需要调用__________方法。

二、选择题

1．__________对象提供对 SQL Server 数据库的连接。

A．SqlConnection
B．SqlCommand
C．SqlDataReader
D．SqlDataAdapter

2．声明连接字符串时，__________表示是否启用连接池。

A．Initial Catalog
B．Database
C．Pooling
D．Server

3．SqlCommand 对象 CommandType 属性的取值不包括__________。

A．Text
B．TableDirect
C．StoredProcedure
D．Index

4．关于 DataSet 与 SqlDataReader 的区别，下面选项错误的是__________。

A．DataSet 不需要和数据库创建连接，而 SqlDataReader 对象必须与数据库进行连接
B．DataSet 处理数据的速度较快，而 SqlDataReader 处理数据的速度较慢
C．DataSet 占用内存较多，而 SqlDataReader 占用内存较少
D．DataSet 支持分页和排序功能，而 SqlDataReader 不支持分页和排序

5．SqlDataAdapter 对象向 DataSet 中填充数据的正确步骤是__________

（1）创建要执行的 SQL 语句或者存储过程。
（2）创建数据库连接对象。

（3）向 DataSet 对象中填充数据。

（4）利用 SQL 语句和数据库连接对象创建 SqlDataAdapter 对象。

A．（2）、（1）、（3）、（4）

B．（2）、（4）、（3）、（1）

C．（2）、（1）、（3）、（4）

D．（2）、（1）、（4）、（3）

6．DataSet、DataTable 和 DataView 之间的关系是__________。

A．DataSet 对象包含 DataTable 对象，DataTable 对象包含 DataView 对象

B．DataSet 对象包含 DataView 对象，DataView 对象又包含 DataTable 对象

C．DataTable 对象同时包含 DataView 对象和 DataSet 对象

D．DataSet 对象同时包含 DataTable 对象和 DataView 对象

三、简答题

1．SqlConnection 对象和 SqlCommand 对象分别是做什么的？

2．DataSet 对象的工作原理是什么？如何使用 SqlDataAdapter 对象向 DataSet 中填充数据？

3.动态创建 Table 对象的一般步骤是什么？

4．请概述 DataSet、DataView 和 DataTable 之间的关系。

第 10 章　数据绑定技术

在使用 ADO.NET 技术连接和操作数据库中的数据时，可以通过 foreach 或者 for 循环语句遍历读取的数据。但是，使用这种方式代码容易出错，而且非常烦琐，不利于维护和修改。ASP.NET 提供了多种控件，其中包括数据控件，使用数据控件只需几行代码便可实现数据绑定。

本章介绍 ASP.NET 的数据绑定技术，首先介绍操作数据时常用的一些绑定方式，然后介绍提供的数据源控件和数据绑定控件。

本章学习要点：

- ❑ 熟悉常见的数据绑定方式
- ❑ 熟悉数据源控件的使用
- ❑ 掌握 Repeater 控件的使用
- ❑ 掌握 DataList 控件的使用
- ❑ 掌握 GridView 控件的使用
- ❑ 了解其他常用的数据绑定控件

10.1　常见的数据绑定

假设当前页面的后台存在一个公有 int 类型的变量，如何将这个变量的值显示到页面中呢？这时可以使用数据绑定技术。ASP.NET 中通常使用<%= %>、<%# %>和<%$ %>三种方式实现绑定。

10.1.1　<%= %>方式绑定

<%= %>绑定数据是最简单的一种绑定方式，它的效果等价于 Response.Write()实现的效果。

【范例 1】

在页面后台声明 string 类型的 userName 变量，并指定该变量的值。代码如下。

```
public string userName = "Administrator";
```

在页面中直接通过<%= %>绑定 userName 变量。代码如下。

```
    直接显示：<%=userName %>
    Label 控件：<asp:Label ID="lbl1" runat="server" Text="Label"><%=userName
%></asp:Label>
```

10.1.2 <%# %>方式绑定

<%# %>是一种最常用的绑定方式，它是控件数据绑定的基础，所有的数据表达式都必须包含在这些字符串中，并且必须调用控件的 DataBind()方法才能正常执行。无论是 Web 服务器控件还是 HTML 服务器控件，都可以通过<%# %>实现绑定。

1．绑定表达式语法

使用<%# %>绑定数据时，数据绑定表达式必须包含在<%#和%>之间。格式如下。

```
<tagprefix : tagname property='<%# data-binding expression %>'
runat="server" />
```

或者：

```
<%# data-binding expression %>
```

ASP.NET 支持分层数据绑定模型，数据绑定表达式使用 Eval()和 Bind()方法将数据绑定到控件，并将更改提交回数据库。

（1）Eval()是静态单向（只读）方法，因此 Eval()方法用于单向（只读）绑定。该方法采用数据字段的值作为参数并将其作为字符串返回。

（2）Bind()方法支持读取和写入功能，因此 Bind()方法用于双向（可更新）绑定。该方法可以检索数据绑定控件的值并将任何更改提交回数据库。

2．数据绑定表达式的出现位置

<%# %>绑定表达式时，可以将数据表达式包含在服务器控件或者普通的 HTML 元素的开始标记中。格式如下。

```
<asp:TextBox ID="TextBox1" runat="server" Text='<%#数据绑定表达式%>' >
</asp:TextBox>
```

上述绑定表达式可以是一个变量，也可以是一个带返回值的 C#或者 VB.NET 方法，还可以是某个控件的某个属性的值，也可以是 C#或者 VB.NET 对象的某个字段或者属性的值等。当然，也可以直接是一个字符串，例如“许飞”。

数据绑定表达式还可以包含在页面中的任何位置。这时，数据绑定表达式的取值可以参考上面。格式如下。

```
<form id="form1" runat="server">
    <div>
        <%#Eval（"数据绑定表达式 1"）%>
        <%#Eval（"数据绑定表达式 2"）%>
    </div>
</form>
```

注意

如果数据绑定表达式是 Eval("数据库中某个表的某个字段")或者 Bind("数据库中某个表的某个字段")等，那么必须把 TextBox 控件放在某个循环显示的控件（如 Repeater 控件）的模型中，否则会提示 Eval()和 Bind()等绑定方法只能在数据绑定控件的上下文中使用。

可以将数据绑定表达式包含在 JavaScript 脚本代码中，从而实现在 JavaScript 中调用 C#或者 VB.NET 的方法。

【范例 2】

单击页面中的按钮调用 JavaScript 中的函数，在函数中调用后台的方法。步骤如下。

（1）创建窗体页面，在页面中添加 Button 控件，并指定 Click 事件。代码如下。

```
<input id="Button1" type="button" value="Javascript 调用 c#的方法!" onclick
="GetStr()" />
```

（2）在 JavaScript 脚本中创建 GetStr()函数，在函数中调用 SayHello()方法并将结果保存到 a 变量中，然后弹出对话框提示。代码如下。

```
function GetStr() {
    var a = '<%# SayHello("小张") %>';          //调用 c#的方法
    alert(a);
}
```

（3）在页面后台添加代码，相关内容如下。

```
public string SayHello(string name) {
    return name + "对夏天说：我喜欢你。";
}
protected void Page Load(object sender, EventArgs e) {
    Page.DataBind();
}
```

（4）运行页面查看效果，单击按钮进行测试，效果图不再显示。

3. 数据绑定表达式的类型

数据绑定表达式的类型有多种，它可以是一个变量，可以是服务器控件的属性值，可以是一个数组等集合对象，可以是一个表达式，可以是一个方法，还可以是使用 Eval()和 Bind()等方法取得的数据表的字段。

【范例 3】

在页面中添加 ListBox 控件，将控件的 DataSource 属性值指定为一个数组集合。代码如下。

```
<asp:ListBox  ID="List1"  DataSource="<%#array  %>"  runat="server">
</asp:ListBox>
```

在页面后台创建 ArrayList 集合对象，调用 Add()方法添加数据。代码如下。

```
public ArrayList array = new ArrayList();
protected void Page Load(object sender, EventArgs e) {
    array.Add("李阳"); array.Add("李濂"); array.Add("许风");
    array.Add("乐乐"); array.Add("陈阳"); array.Add("多多");
    Page.DataBind();
}
```

注 意

如果数据绑定表达式作为属性的值，只要数据绑定表达式中没有出现双引事情，那么<%#数据绑定表达式 %>的最外层用双引号或者单引号都可以。如果数据绑定表达式中出现双引号，则<%#数据绑定表达式 %>的最外层最好用单引号。

10.1.3 <%$ %>方式绑定

<%$ %>用于对 Web.config 文件的键/值进行绑定，通常用于连接数据库的字符串。使用<%$ %>绑定数据时需要注意以下两点。

（1）只能用于绑定 Web 服务器控件。

（2）只能绑定到服务器控件的某个属性上。

【范例 4】

通过 DropDownList 控件的 DataSourceID 属性指定数据源绑定控件，在数据源绑定控件中绑定数据库中的数据。SqlDataSource 控件的代码如下。

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" SelectCommand=
"SELECT [typeId], [typeName] FROM [employType]" ConnectionString="<%$
ConnectionStrings:homeworkConnectionString %>">
```

10.2 数据控件

ASP.NET 中提供了一系列数据控件，如 Chart、GridView 和 DetailsView 控件等。大体来分，可以将最常用的控件分为数据源控件和数据绑定控件。

10.2.1 数据源控件

数据源控件是数据绑定体系结构中的一个关键部分，能够通过数据绑定控件来提供声明性编程模型和自动数据绑定行为。简单来说，数据源控件概括了一个数据存储和可以针对所包含的数据执行的一些操作。数据绑定控件通过 DataSourceID 属性与一个数据源控件相关联。

ASP.NET 中提供的数据源控件允许开发者使用不同类型的数据源，如数据库、XML 文档或者中间层业务对象。如表 10-1 所示为 ASP.NET 中提供的 7 种数据源控件。

表 10-1 常用的数据源控件

数据源控件名称	说明
AccessDataSource	它用于检索 Access 数据库（文件后缀名为.mdb 的文件）中的数据
LinqDataSource	它常常用于访问数据库实体类提供的数据
ObjectDataSource	它能够将来自业务逻辑层的数据对象与表示层中的数据绑定，实现数据的显示、编辑和删除等任务
EntityDataSource	允许绑定到基于实体数据模型（EDM）的数据，支持自动生成更新、插入、删除和选择命令。还支持排序、筛选和分页
SiteMapDataSource	专门处理类似站点地图的 XML 数据，默认情况下数据源为以.sitemap 为扩展名的 XML 文件
SqlDataSource	它可以使用基于 SQL 关系的数据库（如 SQL Server、Oracle、ODBC 以及 OLE DB 等）作为数据源，并从这些数据源中检索数据
XmlDataSource	它常常用来访问 XML 文件或具有 XML 结构层次数据（如 XML 数据块等），并向数据提供 XML 格式的层次数据

在表 10-1 中，SiteMapDataSource、SqlDataSource 和 XmlDataSource 控件最为常用。下面以 SqlDataSource 控件为例，通过范例 5 进行说明。

【范例 5】

读取数据库中 UserInfo 表的全部数据，并将表中 UserName 字段的值绑定到 ListBox 控件。步骤如下。

（1）从工具箱中拖动 ListBox 控件和 SqlDataSource 控件到页面中，并且指定 ListBox 控件的 Width 属性值为 150。

（2）在页面的【设计】窗口中选中 SqlDataSource 控件并单击右上角的小按钮，这时显示 SqlDataSource 任务栏，如图 10-1 所示。



图 10-1 SqlDataSource 控件的任务栏

（3）单击图 10-1 中的【配置数据源】选项弹出【配置数据源】对话框，开发者可以在对话框中选择数据源，也可以单击【新建连接】按钮添加连接，如图 10-2 所示。

（4）在图 10-2 中输入服务器名，然后选择 SQL Server 身份验证，并输入用户名和密码，再选择要连接的数据库，设置完毕后单击【测试连接】按钮进行测试，如图 10-3 所示。

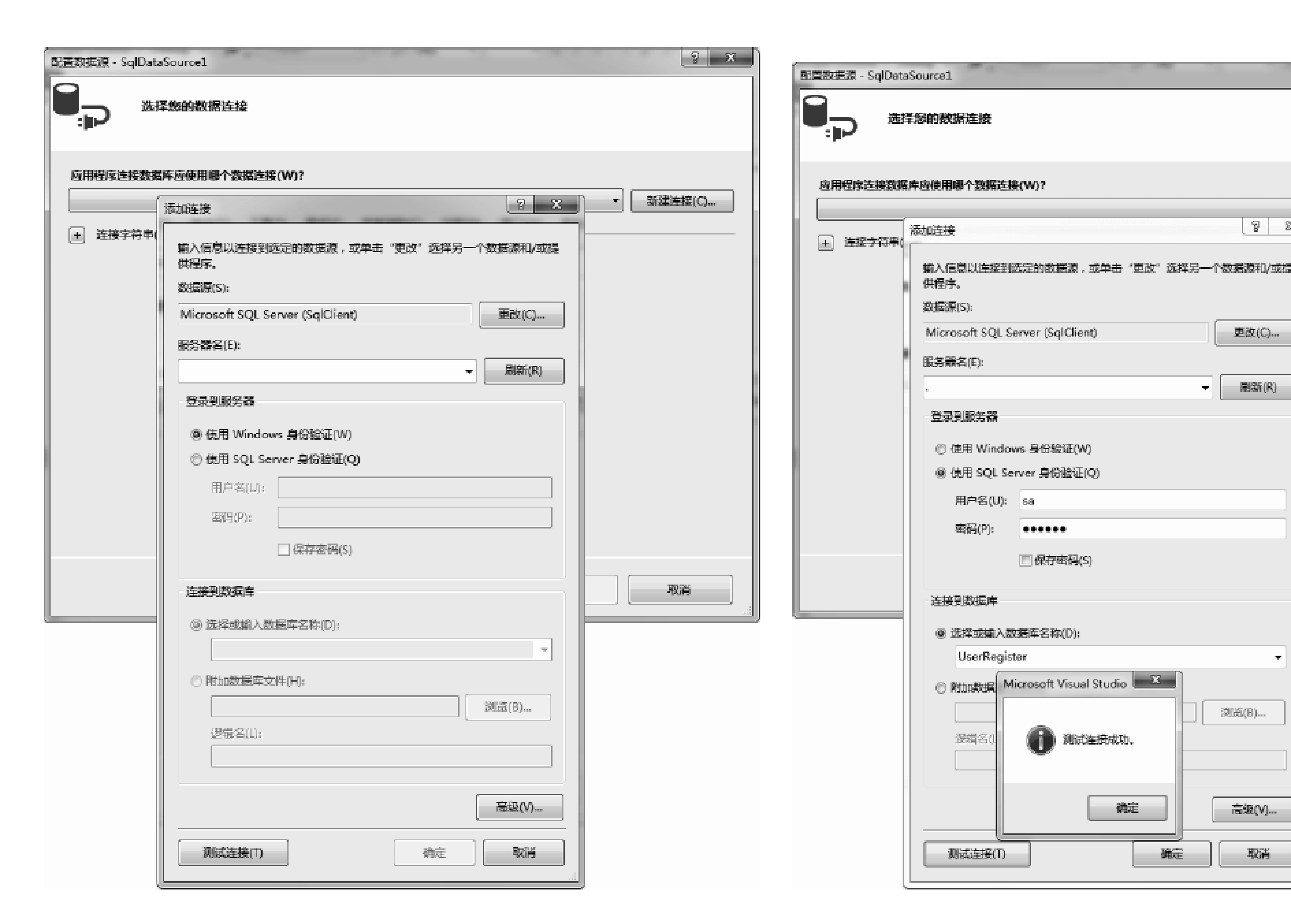


图 10-2　添加连接

图 10-3　测试连接

（5）当显示如图 10-3 所示的“测试连接成功”提示后依次单击【确定】按钮，然后单击【下一步】按钮，弹出如图 10-4 所示的对话框。该对话框提示是否将连接字符串保存到应用程序的配置文件中，默认情况下进行保存，如果不保存，不要选择图中的复选框即可。

（6）单击图 10-4 中的【下一步】按钮执行下一步操作，如图 10-5 所示。在该图中选择要获取的内容，默认情况下选中*，即表示获取所有字段的值，这里只选中 UserName 字段。如果有需要，还可以单击图中的 WHERE 和 ORDER BY 等按钮。

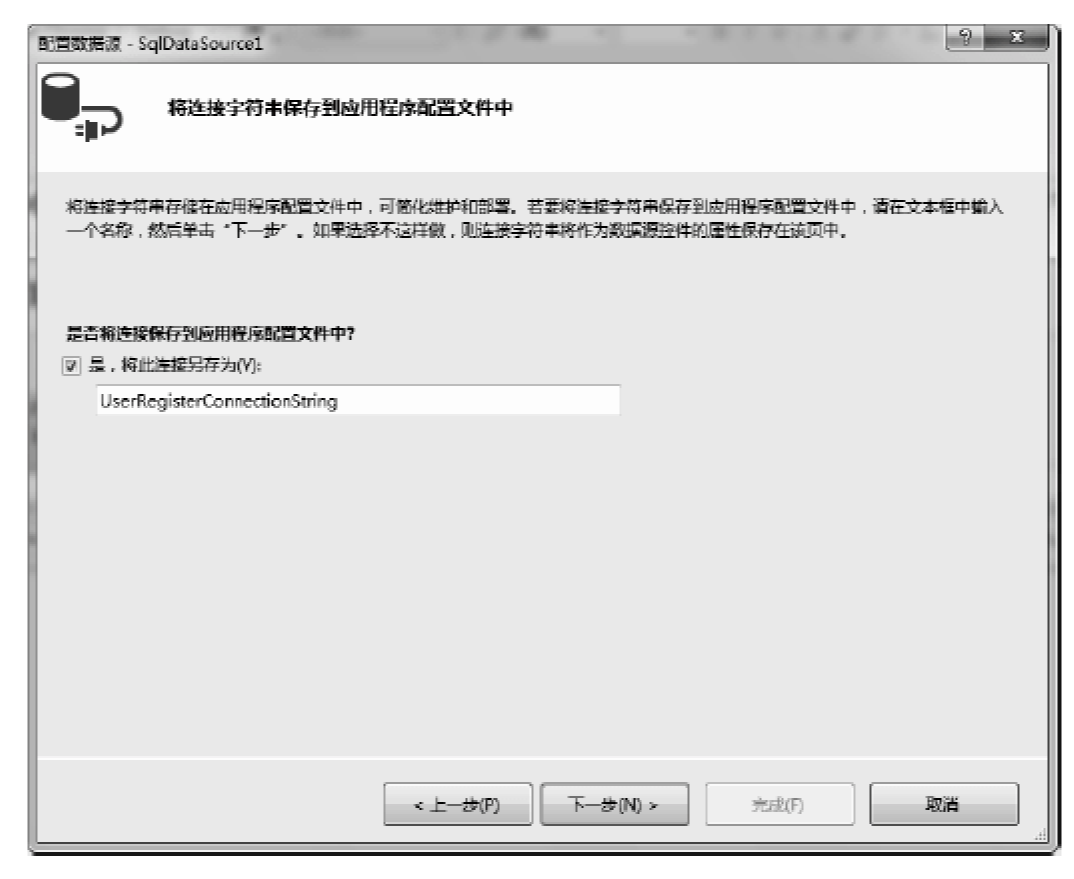


图 10-4　是否保存配置文件



图 10-5　配置 Select 语句

（7）选择完毕后单击图 10-5 中的【下一步】按钮，默认效果如图 10-6 所示。

（8）单击图 10-6 中的【测试查询】按钮查看效果，如图 10-7 所示。



图 10-6 默认测试查询



图 10-7 测试查询效果

（9）单击图 10-7 中的【完成】按钮完成配置，为页面中的 ListBox 控件指定 DataTextField 属性和 DataValueField 属性。代码如下。

```
UserName 字段的集合：<asp:ListBox ID="lbNameList" DataSourceID="Sql
DataSource1" runat="server" Width="150" DataTextField="UserName" DataValue
Field= "UserName"></asp:ListBox><br />
<asp:SqlDataSource ID="SqlDataSource1" runat="server" SelectCommand=
"SELECT [UserName] FROM [UserInfo]" onnectionString="<%$ Connection
Strings:UserRegisterConnectionString %>"></asp:SqlDataSource>
```

（10）运行页面查看效果，如图 10-8 所示。



图 10-8 XmlDataSource 控件的使用

提示

无论是上述案例介绍的 SqlDataSource 控件，还是表 10-1 中的其他数据源控件（如 XmlDataSource 和 SiteMapDataSource），它们的使用方法与 SqlDataSource 控件相似，这里不再一一介绍。

10.2.2 数据绑定控件

数据源控件不能够显示数据，它必须与数据绑定控件一起使用，如前面介绍的 DropDownList、ListBox 和 CheckBoxList 等都是数据绑定控件。除了这些简单的绑定控

件外，ASP.NET 中还提供了一些高级的数据绑定控件，这些控件可以用来显示列表信息与详细信息，如表 10-2 所示列出了一些常见的数据绑定控件。

表 10-2　常见的数据绑定控件

数据绑定控件名称	说明
DataList	以表的形式，通过 DataList 控件可以使用不同的布局来显示数据记录
DetailsView	一次只呈现一条表格形式的记录，并提供翻阅多条记录以及插入、更新和删除记录的功能
FormView	与 DetailsView 控件相似，它一次呈现数据源中的一条记录，并提供翻阅多条记录以及插入和删除的功能。但是它不指定用于显示记录的预定义布局
GridView	以表的形式显示数据，并提供对列进行排序、翻阅数据以及编辑或删除单个记录的功能
ListView	可以使开发者使用模板定义的格式来显示来自数据源的数据
Repeater	使用数据源返回的一组记录呈现只读列表

10.3　Repeater 控件

Repeater 控件是一个数据绑定控件，用于生成各个项的列表。使用模板来定义网页上的各个项的布局，当运行页面时，Repeater 控件为数据源中的每个项重复此布局。一般情况下，将 Repeater 控件称为重复控件。

10.3.1　Repeater 控件的模板

Repeater 控件是容器控件，使用它可以从页的任何可用数据中创建出自定义列表。由于 Repeater 控件没有默认的外观，因此，可以使用 Repeater 控件创建多种列表，如表布局、逗号分隔列表和 XML 格式的列表等。

Repeater 控件不具备内置的呈现功能，开发者必须通过创建模板为 Repeater 控件提供布局。模板可以包含标记和控件的任意组合，如果未定义模板，或者如果模板不包含元素，则当应用程序运行时，Repeater 控件不显示在页面上。

Repeater 控件支持 5 种模板，如表 10-3 所示。

表 10-3　Repeater 控件的 5 种模板

数据绑定控件名称	说明
HeaderTemplate	头部模板。因为数据列表一般会有表头，为保证代码结构化，表头的内容就可以放在这里
ItemTemplate	项目模板。这里就是普通项的模板，也就是数据列表里每一项在页面上展示的效果就在这里定义
AlternatingItemTemplate	交替项模板。对应 ItemTemplate，如果设置该项则表示偶数项的模板，一般设置列表奇偶项不同背景色时会用到
SeparatorTemplate	间隔符模板。在每一个 ItemTemplate 或 AlternatingItemTemplate 项之间插入的分隔用的内容
FooterTemplate	脚注模板。一般的列表项可能会用脚注说明该列表的信息，这里定义列表脚注的内容

【范例 6】

读取 UserInfo 表中的所有记录，这些记录包含 UserName、UserPass、UserMail 和 UserBirth 字段的值。实现步骤如下。

（1）从工具箱中拖动 SqlDataSource 控件到页面中，代码如下。

```
<asp:SqlDataSource ConnectionString="<%$ ConnectionStrings:UserRegister
ConnectionString %>" ID="SqlDataSource1" runat="server" SelectCommand=
"SELECT [UserName], [UserPass], [UserMail],[UserBirth] FROM [UserInfo]">
</asp:SqlDataSource>
```

（2）从工具箱中拖动 Repeater 控件到页面中，代码如下。

```
<asp:Repeater  ID="Repeater1"  runat="server"  DataSourceID="SqlData
Source1">
</asp:Repeater>
```

（3）为 Repeater 控件指定 HeaderTemplate 模板项，代码如下。

```
<HeaderTemplate>
    <table>
        <tr>
            <th style="width:150px;">用户名</th>
            <th style="width:150px;">用户密码</th>
            <th style="width:250px;">电子邮箱</th>
            <th style="width:200px;">出生日期</th>
        </tr>
</HeaderTemplate>
```

（4）为 Repeater 控件指定 ItemTemplate 模板项，在该项中指定每个单元格的背景颜色。代码如下。

```
<ItemTemplate>
    <tr>
        <td style="background-color: #CCFFCC"><asp:Label runat="server"
        ID="lblName" Text='<%# Eval("UserName") %>' /></td>
        <td style="background-color: #CCFFCC"><asp:Label runat="server"
        ID="lblPass" Text='<%# Eval("UserPass") %>' /></td>
        <td style="background-color: #CCFFCC"><asp:Label runat="server"
        ID="lblMail" Text='<%# Eval("UserMail") %>' /></td>
        <td style="background-color: #CCFFCC"><asp:Label runat="server"
        ID="lblBirth" Text='<%# Eval("UserBirth","{0:yyyy 年 MM 月 dd 日}")
        %>' /></td>
    </tr>
</ItemTemplate>
```

（5）为 Repeater 控件指定 AlterItemTemplate 模板项，这里不再指定单元格的背景颜色。代码如下。

```
<AlternatingItemTemplate>
```

```
        <tr>
            <td><asp:Label runat="server" ID="lblName" Text='<%# Eval("User
             Name") %>' /></td>
            <td><asp:Label runat="server" ID="lblPass" Text='<%# Eval("User
             Pass") %>' /></td>
            <td><asp:Label runat="server" ID="lblMail" Text='<%# Eval("User
             Mail") %>' /></td>
            <td><asp:Label runat="server" ID="lblBirth" Text='<%# Eval("User
             Birth","{0:yyyy 年 MM 月 dd 日}") %>' /></td>
        </tr>
    </AlternatingItemTemplate>
```

（6）为 Repeater 控件指定 FooterTemplate 模板项，该模板项包含一个 table 元素的结束标记。

（7）运行页面查看效果，如图 10-9 所示。从该图中可以看出，为 Repeater 控件指定的 ItemTemplate 和 AlterItemTemplate 模板项起到了作用。



图 10-9 Repeater 控件的使用

10.3.2 Repeater 控件的属性

Repeater 控件提供了一系列的属性，如范例 6 为其指定的 DataSourceID 属性。除了这个属性外，Repeater 控件还包含多个常用属性，如表 10-4 所示。

表 10-4 Repeater 控件的常用属性

属性名称	说明
DataSource	获取或设置为填充列表提供数据的数据源
DataSourceID	获取或设置数据源控件的 ID 属性，Repeater 控件应使用它来检索其数据源
Items	获取 Repeater 控件中的 RepeaterItem 对象的集合 RepeaterItemCollection
DataMember	获取或设置 DataSource 中绑定到控件的特定表

在表 10-4 中，Items 属性返回的是一个 RepeaterItemCollection 集合，通过该集合的 Count 属性可以获取设置的列表总数。

【范例 7】

在范例 6 的基础上进行更改，通过指定 DataSource 属性实现 Repeater 控件的绑定。步骤如下。

（1）删除页面中与 SqlDataSource 控件有关的代码。

（2）删除页面中 Repeater 控件的 DataSourceID 属性。

（3）在页面后台的 Load 事件中添加代码，调用 SqlHelper 类的 GetDataSet()方法获取数据库表 UserInfo 中的全部记录，然后指定 Repeater 控件的 DataSource 属性。代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    string sql = "SELECT UserName,UserPass,UserMail,UserBirth FROM User
    Info";
    DataSet ds = SqlHelper.GetDataSet(SqlHelper.connString, CommandType.
    Text, sql, null);
    Repeater1.DataSource = ds;       //指定数据源
    Repeater1.DataBind();            //绑定数据源
}
```

（4）重新运行页面查看效果，效果图不再显示。

10.3.3 Repeater 控件的事件

除了属性外，Repeater 控件还提供了一系列的事件，常见的三个事件分别是 ItemCreated、ItemCommand 和 ItemDataBound。其中，在 Repeater 控件中创建一项时会引发 ItemCreated 事件。

1. ItemCommand 事件

当单击 Repeater 控件中的按钮时会引发 ItemCommand 事件。该事件允许开发者在项模板中嵌入 Button、LinkButton 和 ImageButton 控件。

为 Repeater 控件指定 ItemCommand 事件时需要传入两个参数。代码如下。

```
protected void Repeater1 ItemCommand(object source, RepeaterCommandEvent
Args e) {

}
```

其中，利用 RepeaterCommandEventArgs 对象的 CommandName 和 CommandArgument 可以实现多个操作。CommandName 表示执行命令的名称；CommandArgument 表示执行命令的参数。CommandName 常用的操作命令如表 10-5 所示。

表 10-5　CommandName 的常用操作命令

命令名称	说明
Cancel	取消编辑操作，并将 GridView 控件返回为只读模式
Delete	删除当前记录
Edit	将当前记录置于编辑模式
Sort	对 GridView 控件进行排序
Update	更新数据源中的当前记录
Page	执行分页操作，将按钮的 CommandArgument 属性设置为 First、Last、Next 和 Prev 或页码，以指定要执行的分页操作类型
Select	选择当前记录

【范例 8】

为了方便测试，更改范例 7 中读取的内容，读取 UserInfo 表中所有字段的值，并将 UserID 字段的值显示到页面。然后为项模板添加一个 LinkButton 控件，指定该控件的 CommandName 属性和 CommandArgument 属性。代码如下。

```
<asp:LinkButton  ID="lbDelete"  runat="server"  CommandName="Delete"
CommandArgument='<%#Eval("UserID") %>'>测试删除</asp:LinkButton>
```

为 Repeater 控件添加 ItemCommand 事件，后台中的事件代码如下。

```
protected void Repeater1_ItemCommand(object source, RepeaterCommand
EventArgs e) {
    string name = e.CommandName;
    int value = Convert.ToInt32(e.CommandArgument);
    LinkButton btn = e.Item.FindControl("lbDelete") as LinkButton;
    Page.ClientScript.RegisterStartupScript(this.GetType(),"","<script>
    alert('删除的主键 ID 值是："+value+"')</script>");
}
```

在上述代码中，首先通过 e.CommandName 和 e.CommandArgument 获取命令名称和命令值，然后获取项模板的 LinkButton 控件，最后弹出一个对话框提示。

运行页面查看效果，单击图中的【测试删除】按钮，如图 10-10 所示。



图 10-10　ItemCommand 事件

2. ItemDataBound 事件

在 Repeater 控件中的某一项被数据绑定后但是尚未呈现在页面之前引发 ItemDataBound 事件。大多数情况下，可以在 ItemDataBound 事件中添加自动生成编号的代码。Repeater 控件自动生成编号有以下三种方式。

（1）页面中利用 Container.ItemIndex 属性自动生成编号。直接在项模板中添加该属性，默认值从 0 开始，如果使编号从 1 开始，则需要设置为 Container.ItemIndex+1。代码如下。

```
<ItemTemplate><%#Container.ItemIndex+1 %></ItemTemplate>
<AlterItemTemplate><%#Container.ItemIndex+1 %></AlterItemTemplate>
```

如果需要为 Repeater 控件添加连续编号，执行向下页翻页后序号接前一页的序号时可以用以下代码。

```
<%#Container.ItemIndex+1+(this.AspNetPager.CurrentPageIndex-1)*page
Size%>
```

其中，AspNetPager 表示第三方分页控件 AspNetPager 控件的 ID 属性值，pageSize 是指每页的数据数量。

（2）页面中利用 Repeater 控件的 Items.Count 属性。通过该属性可以获取控件的所有总数据记录。代码如下。

```
<ItemTemplate><%#this.Repeater1.Items.Count+1 %></ItemTemplate>
<AlterItemTemplate><%#this.Repeater1.Items.Count+1 %></AlterItemTemplate>
```

其中，Repeater1 表示 Repeater 控件的 ID 属性值。

（3）页面后台利用 e.Item.ItemIndex 属性。

【范例 9】

观察图 10-10 可以发现，在页面中显示主键 ID 值时，ID 的值并不是连续的。这可能是因为开发者已经删除了主键 ID 值在 5～45 之间的所有记录。下面为 ItemDataBound 事件添加代码实现自动编号。步骤如下。

（1）更改 Repeater 控件中项模板的内容，这里不再显示主键 ID，而是使用自动编号来代替。代码如下。

```
<asp:Label runat="server" ID="lblNumber" Text="0" />
```

（2）为 Repeater 控件指定 ItemDataBound 事件，在事件代码中首先判断当前项是否为数据项，如果是则获取页面中 ID 属性值为 lblNumber 的 Label 控件，并为该控件的 Text 属性赋值。代码如下。

```
protected void Repeater1 ItemDataBound(object sender, RepeaterItemEvent
Args e) {
    if (e.Item.ItemType == ListItemType.Item || e.Item.ItemType == List
    ItemType.AlternatingItem) {
        Label lb = (Label)e.Item.FindControl("lblNumber");
```

```
                                        // ID 为 lblID 的 Label 控件
        lb.Text = Convert.ToString(e.Item.ItemIndex + 1);       //指定值
    }
}
```

（3）运行页面查看效果，如图 10-11 所示。

Repeater控件

localhost:13213/Repeater/RepeaterNumber.aspx

自动编号	用户名	用户密码	电子邮箱	出生日期	操作
1	admin	admin	admin@163.com	1990年12月12日	测试删除
2	zhangsan	zhangsan	zhangsan@163.com	1990年01月01日	测试删除
3	蓝蓝	123456	lanlan@163.com	1988年01月22日	测试删除
4	李非非	123456	lovefei@163.com	1990年05月17日	测试删除
5	Name1	Name1	1@163.com	1990年01月01日	测试删除
6	Name2	Name2	2@163.com	1990年01月01日	测试删除
7	Name3	Name3	3@163.com	1990年01月01日	测试删除
8	Name4	Name4	4@163.com	1990年01月01日	测试删除
9	Name5	Name5	5@163.com	1990年01月01日	测试删除
10	Name6	Name6	6@163.com	1990年01月01日	测试删除

图 10-11 ItemDataBound 事件

10.4 DataList 控件

DataList 控件与 Repeater 控件都属于迭代控件，使用它们可以显示多行单列或者单行多列的数据。但是，DataList 控件要比 Repeater 控件复杂。

10.4.1 DataList 控件的模板

DataList 控件可以用于任何重复结构中的数据，也可以以不同的布局显示行。它提供了 7 种模板，除了具有表 10-3 中 Repeater 控件的模板外，还有以下两种模板。

（1）EditItemTemplate 模板：它是一个编辑项目模板，针对该字段的模板可以设置不同的服务器端控件来处理编辑状态下的不同类型数据。它呈现控件编辑项的内容，应用 EditItemStyle 样式，如果未定义该模板，则使用 ItemTemplate 模板。

（2）SelectedItemTemplate 模板：它为通过使用按钮或其他操作显示选择的数据记录定义布局。其典型用法是提供数据记录的展开视图或用作主/详细关系的主记录，应用 SelectedItemStyle 样式，如果未定义该模板，则使用 ItemTemplate 模板。

开发者可以像 Repeater 控件那样来用 DataList 显示数据库表中的记录。但是，与 Repeater 控件不同的是：DataList 控件的默认行为是在 HTML 表格中显示数据库记录。

10.4.2 DataList 控件的属性

DataList 控件中包含一系列的属性，通过这些属性可以实现不同的操作，常用属性

如表 10-6 所示。

表 10-6 DataList 控件的常用属性

属性名称	说明
DataSource	获取或设置源，该源包含用于填充控件中的项的值列表
DataSourceID	获取或设置数据源控件的 ID 属性，数据列表控件应使用它来检索其数据
EditItemIndex	获取或设置 DataList 控件中要编辑的选定项的索引号，默认值为-1
GridLines	当 RepeatLayout 属性设置为 Table 时，获取或设置 DataList 控件的网格线样式。GridLines 属性的取值说明如下。 （1）None：默认值，网格线什么也不显示 （2）Both：垂直方向和水平方向都显示网格线 （3）Vertical：只在垂直方向上显示网格线 （4）Horizontal：只在水平方向上显示网格线
HorizontalAlign	获取或设置数据列表控件在其容器内的水平对齐方式
Items	获取表示控件内单独项的 DataListItem 对象的集合
RepeatColumns	获取或设置要在 DataList 控件中显示的列数
RepeatDirection	获取或设置 DataList 控件是垂直显示还是水平显示。其值为 Vertical（默认值）和 Horizontal
RepeatLayout	获取或设置控件是在表中显示还是在流布局中显示。其值有 Table（默认值）、Flow、UnorderedList 和 OrderedList
SelectedIndex	获取或设置 DataList 控件中的选定项的索引
SelectedItem	获取或设置 DataList 控件中的选定项
SelectedValue	获取所选择的数据列表项的键字段的值
ShowHeader	获取或设置一个值，该值指示是否在 DataList 控件中显示页眉节。默认值为 true
ShowFooter	获取或设置一个值，该值指示是否在 DataList 控件中显示脚注部分。默认值为 true

【范例 10】

通过 DataList 控件显示 UserInfo 表中的数据，并且通过表 10-6 中的 GridLines、RepeatColumns 和 RepeatDirection 属性进行设置。步骤如下。

（1）在页面中添加 DataList 控件，该控件的 ID 属性值为 DataList1。DataList 控件包含一个 ItemTemplate 模板，代码如下。

```
<asp:DataList ID="DataList1" runat="server">
    <ItemTemplate>
        <div style="border: 2px solid turquoise; margin:2px;">
            <table>
                <tr><td>主键 ID：<asp:Label runat="server" ID="lblID" Text=
                '<%# Eval("UserID") %>' /></td><td>用户名：<asp:Label runat=
                "server" ID="lblName" Text='<%# Eval("UserName") %>' />
                </td><td>密码：<asp:Label runat="server" ID="lblPass" Text=
                '<%# Eval("UserPass") %>' /></td></tr>
                <tr><td colspan="3">邮箱：<asp:Label runat="server" ID=
                "lblMail" Text='<%# Eval("UserMail") %>' /></td></tr>
                <tr><td colspan="3">出生日期：<asp:Label runat="server" ID=
                "lblBirth" Text='<%# Eval("UserBirth","{0:yyyy 年 MM 月 dd
                日}") %>' /></td></tr>
                <tr><td colspan="3">个人说明：<asp:Label runat="server"
```

```
                ID="lblIntro" Text='<%# Eval("UserIntro") %>' /></td></tr>
            </table>
        </div>
    </ItemTemplate>
</asp:DataList>
```

（2）在页面后台的 Load 事件中添加代码，通过 DataSource 属性指定数据源。代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    string sql = "SELECT * FROM UserInfo";
    DataSet ds = SqlHelper.GetDataSet(SqlHelper.connString, CommandType.
    Text, sql, null);
    DataList1.DataSource = ds;       //指定数据源
    DataList1.DataBind();            //绑定数据源
}
```

（3）运行页面查看效果，如图 10-12 所示。

（4）为 DataList 控件指定 GridLines 属性，将属性值设置为 Both。然后刷新页面，如图 10-13 所示。

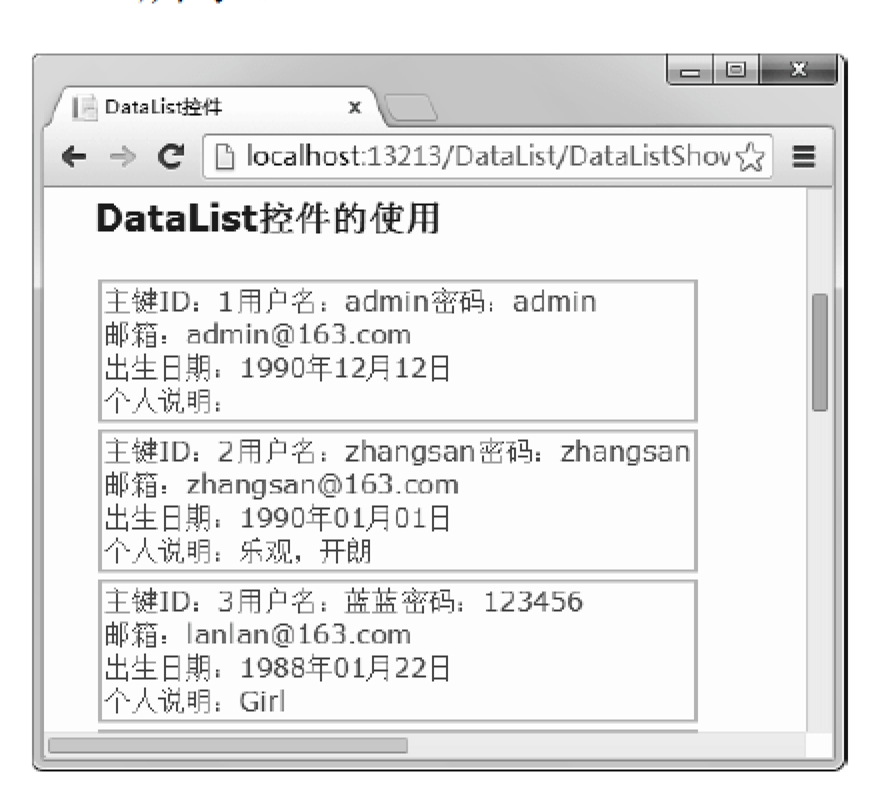


图 10-12　页面默认效果

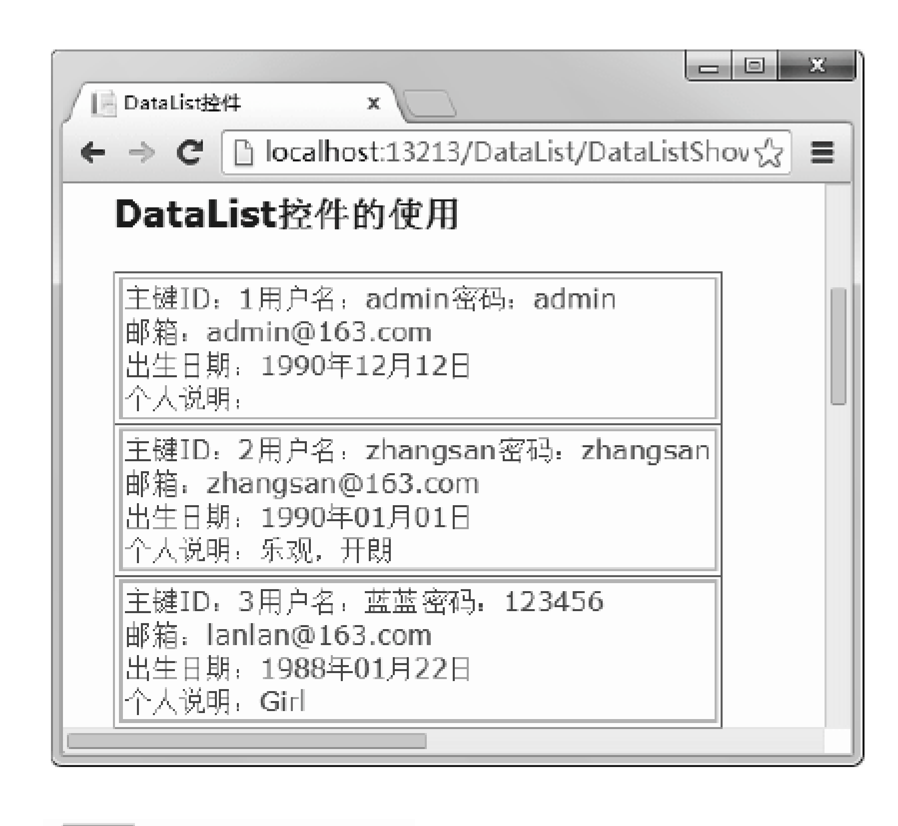


图 10-13　设置 GridLines 属性

（5）为 DataList 控件添加 RepeatColumns 属性，将属性值设置为 2。然后刷新页面，如图 10-14 所示。



图 10-14　RepeatColumns 属性值为 2

（6）为 DataList 控件添加 RepeatDirection 属性，将属性值设置为 Horizontal。然后刷新页面，如图 10-15 所示。



图 10-15　**RepeatDirection 属性值为 Horizontal**

10.4.3　DataList 控件的事件

DataList 控件提供一系列的事件，它的事件要比 Repeater 多，常用事件如表 10-7 所示。

表 10-7　**DataList 控件的常用事件**

事件名称	说明
DataBinding	当服务器控件绑定到数据源时发生
DeleteCommand	对 DataList 控件中的某项单击 Delete 按钮时发生
EditCommand	对 DataList 控件中的某项单击 Edit 按钮时发生
ItemCommand	当单击 DataList 控件中的任一按钮时发生
ItemCreated	当在 DataList 控件中创建项时在服务器上发生
ItemDataBound	当项被数据绑定到 DataList 控件时发生
SelectedIndexChanged	在两次服务器发送之间，在数据列表控件中选择了不同项时发生
UpdateCommand	对 DataList 控件中的某项单击 Update 按钮时发生

在表 10-7 中，如果要引发 CancelCommand、EditCommand、DeleteCommand 和 UpdateCommand 事件，可以将 Button、LinkButton 或 ImageButton 控件添加到 DataList 控件的模板中，并将这些按钮的 CommandName 属性设置为某个关键字，如 Delete。当用户单击项中的某个按钮时就会向该按钮的容器（DataList 控件）发送事件，按钮具体引发哪个事件将取决于所单击按钮的 CommandName 属性的值。

【范例 11】

在范例 10 的基础上添加新的代码，实现自动生成编号的功能。步骤如下。

（1）在页面 DataList 控件的 ItemTemplate 模板的表格中添加一行，代码如下。

```
<tr><td colspan="3">自动编号：<asp:Label ID="lblNumber" runat="server">
</asp:Label></td></tr>
```

（2）为 DataList 控件添加 ItemDataBound 事件，在事件代码中判断对象的索引是否为–1，如果不是获取当前索引项，并将当前索引项加 1，然后将索引值绑定到 ID 属性值为 lblNumber 的 Label 控件中。代码如下。

```
protected  void  DataList1_ItemDataBound(object  sender,  DataListItem
EventArgs e) {
    if (e.Item.ItemIndex != -1) {                    //如果索引不是-1
        int id = e.Item.ItemIndex;                   //当前索引项
        ((Label)e.Item.FindControl("lblNumber")).Text = (id + 1).ToString();
    }
}
```

（3）运行页面查看效果，如图 10-16 所示。



图 10–16　DataList 控件的 ItemDataBound 事件

10.4.4　自动套用格式

DataList 控件提供一系列的属性，用户通过指定相关的样式属性可以定义显示的样式，但是直接选择自定义的样式更加方便。为 DataList 控件自套用格式很简单，在【设计】窗口中选中 DataList 控件，然后单击右上角的按钮显示该控件的任务，单击【自动套用格式】选项弹出如图 10-17 所示的对话框。



图 10–17　自动套用格式

【范例 12】

在图 10-17 中，开发者可以选择列出的一些样式架构。如选择名称为“雨天”的样式架构后单击【确定】按钮，此时在页面的 DataList 控件中新增一些样式代码。内容如下。

```
<AlternatingItemStyle BackColor="#DCDCDC" />
<FooterStyle BackColor="#CCCCCC" ForeColor="Black" />
<HeaderStyle BackColor="#000084" Font-Bold="True" ForeColor="White" />
<ItemStyle BackColor="#EEEEEE" ForeColor="Black" />
<SelectedItemStyle  BackColor="#008A8C"  Font-Bold="True"  ForeColor=
"White" />
```

重新运行或刷新页面，效果如图 10-18 所示。



图 10-18　**DataList 控件自动套用格式**

10.5 实验指导——PagedDataSource 类实现分页

无论是 Repeater 控件还是 DataList 控件，它们都不内置分页功能。实现分页功能时，开发者需要通过其他方式实现，如使用第三方的 AspNetPager 控件，借助于 PagedDataSource 类，或者是通过分页存储过程等。本节实验指导利用 PagedDataSource 类对 Repeater 控件实现分页。

实现页面代码之前，需要首先创建 BookInfo 数据库表，该表包含 BookIsbn、BookTitle、BookAuthor、BookPubdate、BookPublish、BookIntro 和 BookImage 等字段，字段说明如表 10-8 所示。

表 10-8　**BookInfo 表的字段说明**

字段名称	类型	是否为主键	是否为自动增长列	说明
BookIsbn	nvarchar(50)	是	否	图书编号
BookTitle	nvarchar(50)	否	否	图书标题
BookAuthor	nvarchar(20)	否	否	图书作者
BookPubdate	date	否	否	发布日期
BookPublish	nvarchar(50)	否	否	出版社。默认值为“暂无”
BookIntro	text	否	否	图书简介。默认值为“暂无”
BookImage	nvarchar(50)	否	否	图片

创建数据库表完毕后需要向表中添加记录，如图 10-19 所示。



SJB.UserRegister - dbo.BookInfo

	BookIsbn	BookTitle	BookAuthor	BookPubdate	BookPublish	BookIntro	BookImage
▶	ISBN20110001	致我们终将逝去的青春	辛夷坞	2011-07-01	暂无	暂无	~/ShiYan1/Images/im...
	ISBN20110002	你若安好 便是晴天	白落梅	2011-09-01	暂无	暂无	~/ShiYan1/Images/im...
	ISBN20110003	随遇而安	孟非	2011-10-01	暂无	《随遇而安》...	~/ShiYan1/Images/im...
	ISBN20120002	谁的青春不迷茫	刘同	2012-07-02	暂无	暂无	~/ShiYan1/Images/im...
	ISBN20120003	活着（精）	余华	2012-09-01	作家出版社	讲述了农村人...	~/ShiYan1/Images/im...
	ISBN20130001	繁花	金宇澄	2013-04-27	上海文艺出版社	这是一部地域...	~/ShiYan1/Images/im...
	ISBN20130002	平凡的世界	路遥	2013-12-01	北京十月文艺出版社	这是一部现实...	~/ShiYan1/Images/im...
	ISBN20130003	北京，北京	冯唐	2013-09-01	天津人民出版社	《北京，北京...	~/ShiYan1/Images/im...
	ISBN20130004	万物生长	冯唐	2013-09-01	天津人民出版社	《万物生长》...	~/ShiYan1/Images/im...
	ISBN20140001	妈阁是座城	严歌苓	2014-01-01	人民文学出版社	暂无	~/ShiYan1/Images/im...
	ISBN20140002	我这辈子有过你	张小娴	2014-03-01	湖南文艺出版社	讲述了三个始...	~/ShiYan1/Images/im...
	ISBN20140003	陆犯焉识（新版）	严歌苓	2014-05-01	暂无	陆焉识本是上...	~/ShiYan1/Images/im...
	ISBN20140004	我在云上爱你	张小娴	2014-05-01	人民文学出版社	先喜欢上你，...	~/ShiYan1/Images/im...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

1 / 13

图 10–19　BookInfo 表中的数据记录

设计页面并在后台添加实现代码，步骤如下。

（1）在页面中添加 Repeater 控件，为该控件添加 ItemTemplate 模板，在项模板中绑定数据库中的数据。代码如下。

```
<asp:Repeater ID="Repeater1" runat="server">
    <ItemTemplate>
        <table>
            <tr><td rowspan="4"><asp:Image runat="server" ImageUrl='<%#Eval
            ("BookImage") %>' Width="120" Height="150" /></td></tr>
            <tr><td style="font-size:16px; color:blue; font-weight:bold;">
            <%#Eval("BookTitle") %></td></tr>
            <tr><td>作者：<%#Eval("BookAuthor") %>   出版社：
            <%#Eval("BookPublish") %>   出版日期：<%#Eval
            ("BookPubdate","{0:yyyy 年 MM 月 dd 日}") %></td></tr>
            <tr><td><%#Eval("BookIntro") %></td></tr>
        </table>
    </ItemTemplate>
</asp:Repeater>
```

（2）在 Repeater 控件之后添加一个 Label 控件和两个 Button 控件，Label 控件显示当前页和总页数，Button 控件分别执行上一页和下一页操作。代码如下。

```
<asp:Label ID="Label1" runat="server" Text=""></asp:Label>  
  
<asp:Button ID="btnPrev" class="btn1 mouseout" runat="server" Text="上
一页" OnClick="btnPrev Click" />
<asp:Button ID="btnNext" class="btn1_mouseout" runat="server" Text="下
一页" OnClick="btnNext_Click" />
```

（3）在页面后台添加代码，首先声明一个公有的 Pager 变量，通过 ViewState 对象保存 Page 变量的值。代码如下。

```
public int Pager {
```

```
        get { return (int)ViewState["Page"]; }
        set { ViewState["Page"] = value; }
    }
```

（4）添加 ListBinding()方法，该方法通过 PagedDataSource 实现分页，并且将数据绑定到 Repeater 控件。代码如下。

```
    public void ListBinding() {
        PagedDataSource pds = new PagedDataSource();
        pds.DataSource = SqlHelper.GetDataSet(SqlHelper.connString, Command
        Type.Text, "SELECT * FROM BookInfo", null).Tables[0].DefaultView;
        pds.AllowPaging = true;             //允许分页
        pds.PageSize = 2;                   //每页显示两条记录
        pds.CurrentPageIndex = Pager;       //当前页的索引
        btnPrev.Enabled = true;             //【上一页】按钮可用
        btnNext.Enabled = true;             //【下一页】按钮可用
        if (pds.IsFirstPage)                //如果是首页，【上一页】按钮不可用
            btnPrev.Enabled = false;
        if (pds.IsLastPage)                 //如果是尾页，【下一页】按钮不可用
            btnNext.Enabled = false;
        Label1.Text = "第" + (pds.CurrentPageIndex + 1).ToString() + "页 共"
       + pds.PageCount.ToString() + "页";
        Repeater1.DataSource = pds;         //指定 Repeater 控件的数据源
        Repeater1.DataBind();               //绑定到 Repeater 控件
    }
```

在上述代码中，首先创建 PagedDataSource 类的实例对象 pds，接着指定该对象的 DataSource、AllowPaging、PageSize 和 CurrentPageIndex 属性，然后分别调用 IsFirstPage 和 IsLastPage 判断当前显示页是否为首页和尾页，最后设置 Label 控件的值，并绑定数据源。

（5）为页面中的 Load 事件添加代码，首先判断页面是否首次加载，如果是则将 Page 的值设置为 0，并且调用 ListBinding()方法绑定数据。代码如下。

```
    protected void Page_Load(object sender, EventArgs e) {
        if (!IsPostBack) {              //如果是首页加载
            ViewState["Page"] = 0;
            ListBinding();              //绑定数据
        }
    }
```

（6）为页面中【上一页】和【下一页】按钮添加 Click 事件代码，以【上一页】按钮为例，单击该按钮时将 Pager 的索引值减 1（如果是【下一页】按钮，则加 1），然后调用 ListBinding()方法重新绑定数据。代码如下。

```
    protected void btnPrev_Click(object sender, EventArgs e) {
        Pager--;                        //当前页索引-1
        ListBinding();                  //重新绑定页面
    }
```

（7）运行页面查看效果，如图 10-20 所示。



图 10-20 页面实现的分页效果

（8）单击图 10-20 中的【下一页】按钮进行测试，效果图不再显示。

PagedDataSource 类中包含多个属性，前面步骤中已经使用到部分属性，如表 10-9 所示对常用属性进行说明。

表 10-9 PagedDataSource 类的常用属性

属性名称	说明
AllowCustomPaging	获取或设置一个值，指示是否在数据绑定控件中启用自定义分页
AllowPaging	获取或设置一个值，指示是否在数据绑定控件中启用分页
AllowServerPaging	获取或设置一个值，指示是否启用服务器端分页
Count	获取要从数据源使用的基数
CurrentPageIndex	获取或设置当前页的索引
DataSource	获取或设置数据源
DataSourceCount	获取数据源中的项数
FirstIndexInPage	获取页面中显示的首条记录的索引
IsFirstPage	获取一个值，该值指示当前页是否是首页
IsLastPage	获取一个值，该值指示当前页是否是最后一页
IsPagingEnabled	获取一个值，该值指示是否启用分页
PageCount	获取显示数据源中的所有项所需要的总页数
PageSize	获取或设置要在单页上显示的项数

10.6 GridView 控件

在 ASP.NET 提供的几种高级数据绑定控件中，Repeater、DataList 和 GridView 是最常使用的三个控件。与前两种控件相比，GridView 控件的功能要更加强大，也更加复杂。

10.6.1 GridView 控件的功能

GridView 控件通常以表格的形式显示数据，通常被称为网格视图控件。GridView 控件最常用的一种形式是后台管理系统，显示对数据库中的操作。GridView 控件的功能非常强大，主要体现在以下几点。

（1）以编程方式访问 GridView 对象模型以动态设置属性、处理事件等。

（2）可通过主题和样式进行自定义的外观。

（3）绑定到数据源控件，如 SqlDataSource 和 ObjectDataSource 等。

（4）内置排序和分页功能。

（5）内置更新和删除功能。

（6）内置行选择功能。

（7）用于超链接列的多个数据字段。

（8）多个键字段。

10.6.2 GridView 控件的模板

GridView 控件自身提供两个模板：一个是数据模板 EmptyDataTemplate；一个是分页模板 PagerTemplate。

（1）EmptyDataTemplate 模板：当 GridView 控件的数据源为空时将显示该模板的内容。

（2）PagerTemplate 模板：分页模板，定义与 GridView 控件的页导航相关的内容。

在实际操作过程中，数据模板和分页模板并不经常使用。最常用的是 GridView 提供的 TemplateField 字段中的 6 个模板，如表 10-10 所示。

表 10-10　GridView 控件 TemplateField 字段的模板

模板名称	说明
HeaderTemplate	头部模板，设置每一列头部的提示内容及格式
FooterTemplate	脚注模板，设置每一列底部的提示内容及格式
ItemTemplate	项目模板，设置列内容的格式
AlternatingTemplate	交替项，使奇数条数据及偶数条数据以不同的模板显示，该模板与 ItemTemplate 结合可产生两个模板交错显示的效果
InsertItemTemplate	数据添加模板
EditItemTemplate	编辑项目模板，这里针对该字段的模板，可以设置不同的服务器端控件来处理编辑状态下的不同类型数据

如下代码为 TemplateField 字段中的模板使用语法。

```
<asp:GridView ID="GridView1" runat="server">
    <Columns>
        <asp:TemplateField>
            <HeaderTemplate><%--头部内容--%></HeaderTemplate>
```

```
            <InsertItemTemplate><%-- 插入模板内容 --%></InsertItemTemplate>
            <EditItemTemplate><%--编辑模板内容--%></EditItemTemplate>
            <ItemTemplate><%--项模板--%></ItemTemplate>
            <AlternatingItemTemplate><%--交替项模板--%></AlternatingItem
            Template>
            <FooterTemplate><%--底部模板--%></FooterTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

10.6.3 GridView 控件的字段

GridView 控件的每一列都由一个 DataControlField 对象表示。默认情况下，GridView 控件的 AutoGenerateColumns 属性的值为 true，表示为数据源中的每一个字段创建 AutoGeneratedField 对象。然后，每一个字段按照在数据源中出现的顺序在 GridView 控件中呈现为一个列。

如果将 AutoGenerateColumns 属性的值设置为 false，开发者可以自定义字段列集合，也可以手动控制哪些字段显示在 GridView 控件中。GridView 控件提供了多种字段类型，下面简单进行介绍。

1. BoundField 字段

BoundField 显示数据源中某个字段的值，这是 GridView 控件的默认字段。一般情况下，使用 BoundField 显示普通文本，常用属性如表 10-11 所示。

表 10-11 **BoundField 字段的常用属性**

属性名称	说明
HeaderText	设置显示的标头文本
DataField	设置绑定到数据源中的列
SortExpression	设置与字段关联的排序表达式
HtmlCode	表示字段是否以 HTML 编码的形式显示给用户。默认值为 true
DataFormatString	设置显示的格式，常用格式如下。 （1）{0:C}：设置要显示的内容是货币类型 （2）{0:D}：设置显示的内容是数字 （3）{0:yy-mm-dd}：设置显示的是日期格式

注 意

使用 DataFormatString 属性设置显示内容的格式时，必须将 HtmlCode 属性的值设置为 false，否则 DataFormatString 的设置无效。

2. CheckBoxField 字段

CheckBoxField 表示复选框字段，以复选按钮的形式显示数据，一般用于展示布尔类型的数据。

3. HyperLinkField 字段

HyperLinkField 将数据源中某个字段的值显示为超链接，它使开发者可以将另一个字段绑定到超链接的 URL。

4. ImageField 字段

ImageField 为 GridView 控件中的每一项显示一个图像，这里一般显示为缩略图片。

5. ButtonField 字段

ButtonField 为 GridView 控件中的每个项显示一个命令按钮。这使开发者可以创建一列自定义按钮控件，例如“添加”按钮或“移除”按钮。

6. CommandField 字段

CommandField 用来执行选择、编辑或者删除操作的预定义命令按钮。

7. TemplateField 字段

TemplateField 指定的模板为 GridView 控件中的每一项显示用户定义的内容。它使开发者可以创建自定义的列字段。与前面几种字段相比，TemplateField 是最灵活的绑定形式，同时它也是最为复杂的。

GridView 控件添加字段的常用方式是：选中 GridView 控件后在【属性】窗格中打开 Columns 属性，单击该属性后的图标弹出【字段】对话框，如图 10-21 所示。

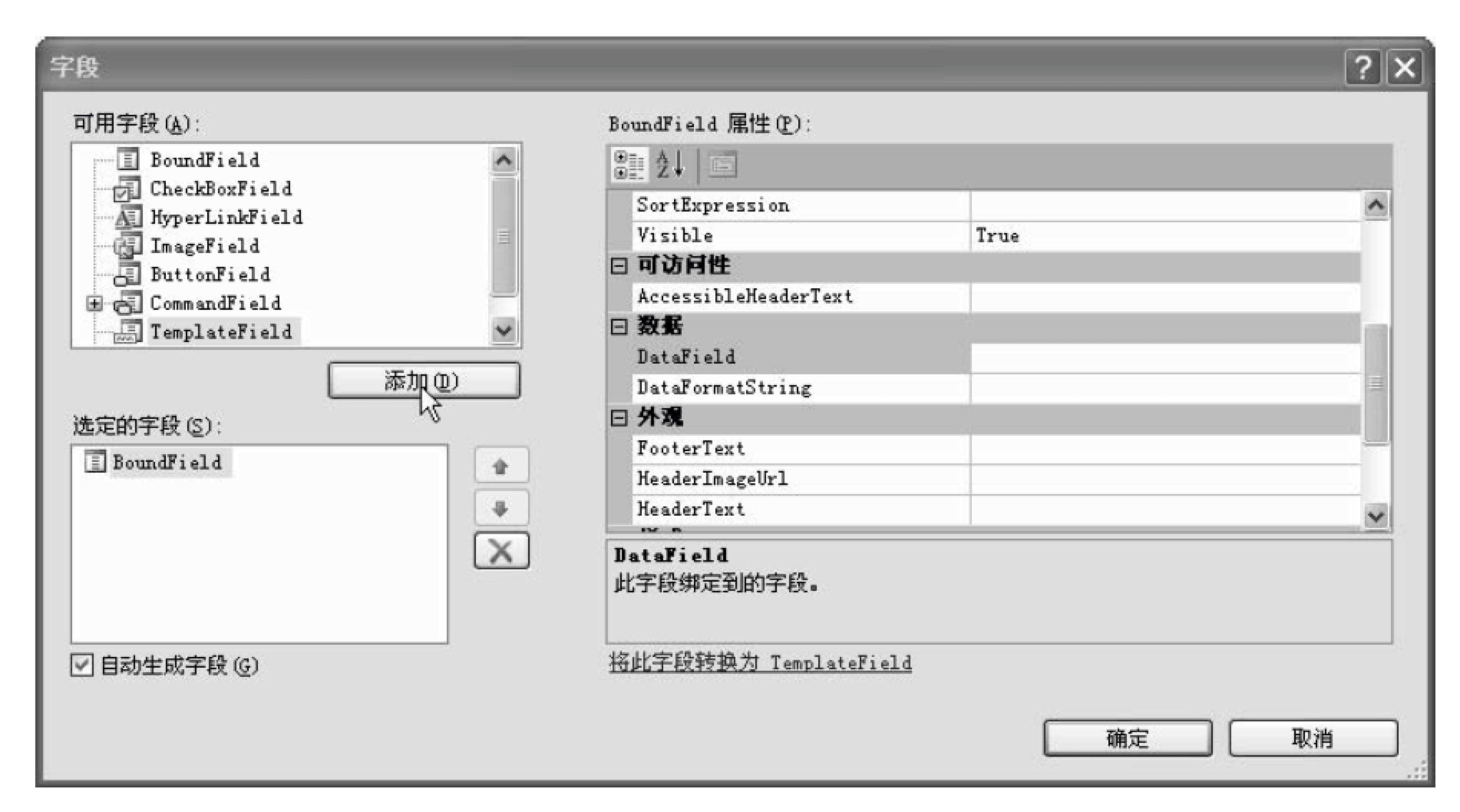


图 10-21 GridView 控件添加字段

在图 10-21 中，开发者选择可用字段后单击【添加】按钮，这时添加的内容会显示在选定的字段中。然后，在右侧可以设置字段的相关属性，设置完毕后单击【确定】按钮。

10.6.4 GridView 控件的属性

GridView 控件包含数十个属性，可以将这些属性分为基本属性、分页属性、样式属性以及其他属性等。下面简单了解 GridView 控件的基本属性和分页属性。

1．基本属性

基本属性是指 GridView 控件最常用使用的一些属性，如表 10-12 所示。

表 10-12 GridView 控件的基本属性

属性名称	说明
AllowSorting	获取或设置一个值，该值指示是否启用排序功能。默认为 false
AutoGenerateColumns	获取或设置一个值，该值指示是否为数据源中的每个字段自动创建绑定字段。默认为 true
AutoGenerateDeleteButton	获取或设置一个值，该值指示是否为每个数据行添加“删除”按钮。默认为 false
AutoGenerateEditButton	获取或设置一个值，该值指示是否为每个数据行添加“编辑”按钮。默认为 false
AutoGenerateSelectButton	获取或设置一个值，该值指示是否为每个数据行添加“选择”按钮。默认为 false
CellSpacing	获取或设置单元格间的空间量
CellPadding	获取或设置单元格的内容和单元格的边框之间的空间量
Columns	获取表示该控件中列字段的 DataControlField 集合
DataMember	当数据源包含多个不同的数据项列表时，获取或设置数据绑定控件到的数据列表名称
DataKeyNames	获取或设置一个数组，该数组包含显示在 GridView 控件中项的主键字段的名称
DataKeys	获取一个 DataKey 集合，这些对象表示 GridView 控件中的每一行的数据键值
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DataSourceID	获取或设置控件的 ID，数据绑定控件从控件中检索其数据项列表
EditIndex	获取或设置要编辑的行的索引
EmptyDataText	获取或设置 GridView 控件绑定到不包含任何记录数据源时所呈现的空数据行中显示的文本
GridLines	获取或设置 GridView 控件的网格线样式，取值说明如下。 （1）Both：同时呈现水平和垂直网格线 （2）Horizontal：仅呈现水平网格线 （3）None：不呈现网格线 （4）Vertical：仅呈现垂直网格线
HorizontalAlign	获取或设置 GridView 控件在页面上的水平对齐方式
Rows	获取表示该控件中数据行中 GridViewRow 对象的集合
SelectedIndex	获取或设置 GridView 控件中选中行的索引
SelectedValue	获取 GridView 控件中选中行的数据键值
SelectedDataKey	获取 DataKey 对象，该对象包含 GridView 控件中选中行的数据键值

续表

属性名称	说明
SelectedRow	获取对 GridViewRow 对象的引用，该对象表示控件中的选中行
SortDirection	获取正在排序的列的排序方向
SortExpression	获取与正在排序的列关联的排序表达式

【范例 13】

利用 GridView 控件显示 BookInfo 表中的数据，步骤如下。

（1）在页面中添加一个 GridView 控件，代码如下。

```
<asp:GridView ID="GridView1" runat="server"></asp:GridView>
```

（2）在页面后台的 Load 事件中添加代码，通过 SqlHelper 类的 GetDataSet()方法获取 BookInfo 表中的数据，并将结果绑定到 GridView 控件。代码如下。

```
protected void Page_Load(object sender, EventArgs e) {
    GridView1.DataSource = SqlHelper.GetDataSet(SqlHelper.connString,
    CommandType.Text, "SELECT * FROM BookInfo", null);
    GridView1.DataBind();
}
```

（3）运行页面查看效果，如图 10-22 所示。从图 10-22 中可以看出，默认情况下 AutoGenerateColumns 属性的值为 true，会自动为 GridView 控件生成字段列。



图 10-22　GridView 控件自动生成字段

（4）默认情况下，GridView 控件的 GridLines 属性的值为 Both，更改该属性的值，将其设置为 Vertical。然后刷新页面查看效果，如图 10-23 所示。

（5）从图 10-22 和图 10-23 中可看出，自动生成列时会将所有的字段都显示出来。但是有些数据是不需要显示的，如 BookPublish 字段中的数据；有些数据是需要经过处理的，如 BookPubdate 和 BookImage 中的数据。更改 GridView 控件的代码，将该控件的 AutoGenerateColumns 属性的值设置为 false，GridLines 属性的值设置为 Both。



图 10–23　指定 GridView 控件的 Both 属性

（6）在 GridView 控件的 Columns 中添加数据，通过 BoundField 字段显示数据。代码如下。

```
<asp:GridView ID="GridView1" runat="server" GridLines="Vertical" Width=
"100%" AutoGenerateColumns="false">
    <Columns>
        <asp:BoundField DataField="BookIsbn" HeaderText="编号" ItemStyle
        -Width="100" ItemStyle-ForeColor="Red" />
        <asp:BoundField DataField="BookTitle" HeaderText="标题" ItemStyle
        -Width="140" />
        <asp:BoundField DataField="BookAuthor" HeaderText="作者" Item
        Style-Width="80" ItemStyle-ForeColor="Blue" />
        <asp:BoundField DataField="BookPubdate" HeaderText="出版日期"
       DataFormat String="{0:yyyy-MM-dd}" ItemStyle-Width="100" />
        <asp:BoundField DataField="BookIntro" HeaderText="简介说明" />
        <asp:ImageField DataImageUrlField="BookImage" HeaderText="封面图
        片">
            <ControlStyle Height="150px" Width="120px" />
        </asp:ImageField>
    </Columns>
</asp:GridView>
```

（7）运行页面查看效果，如图 10-24 所示。



图 10–24　为 GridView 控件自定义字段

2. 分页属性

GridView 控件内置分页功能，实现分页时需要设置一些属性，如表 10-13 所示。

表 10-13 GridView 控件的分页属性

属性名称	说明
AllowPaging	获取或设置一个值，该值指示是否启用分页功能。默认为 false
PageCount	获取在 GridView 控件中显示数据源记录所需的页数
PageIndex	获取或设置当前显示页的索引
PageSize	获取或设置 GridView 控件在每页上所显示的记录条数
PagerSettings	设置 GridView 控件中页导航按钮的属性

在表 10-13 中，PagerSettings 属性返回一个 PagerSettings 对象，使用这个对象可以设置 GridView 控件中的页导航按钮的属性。PagerSettings 对象的常用属性如表 10-14 所示。

表 10-14 PagerSettings 对象的常用属性

属性名称	说明
FirstPageImageUrl	获取或设置为【第一页】按钮显示的图像的 URL
FirstPageText	获取或设置为【第一页】按钮显示的文字
LastPageImageUrl	获取或设置为【最后一页】按钮显示的图像的 URL
LastPageText	获取或设置为【最后一页】按钮显示的文字
Mode	获取或设置分页的控件中的页导航控件的显示模式
NextPageImageUrl	获取或设置为【下一页】按钮显示的图像的 URL
NextPageText	获取或设置为【下一页】按钮显示的文字
PreviousPageImageUrl	获取或设置为【上一页】按钮显示的图像的 URL
PreviousPageText	获取或设置为【上一页】按钮显示的文字

在表 10-14 中，Mode 属性的取值是 PagerButtons 的值之一，取值说明如下。

（1）NextPrevious：一组由【上一页】和【下一页】按钮组成的分页控件。

（2）NextPreviousFirstLast：一组由【上一页】、【下一页】、【首页】和【尾页】按钮组成的分页控件。

（3）NumericFirstLast：一组由带编号的链接按钮以及【首页】和【尾页】链接按钮组成的分页控件。

（4）Numeric：默认值，一组由用户直接访问页的带编号的链接按钮组成的分页控件。

【范例 14】

在范例 13 的基础上添加代码，指定 GridView 控件 AllowPaging 属性的值为 true，PageSize 属性的值为 2。代码如下。

```
<asp:GridView ID="GridView1" runat="server" Width="100%" AutoGenerate
Columns="False" AllowPaging="true" PageSize="2"></asp:GridView>
```

运行页面查看效果，如图 10-25 所示。



图 10-25 GridView 控件实现分页 1

指定 GridView 控件的 PagerSettings 属性，代码如下。

```
<PagerSettings  FirstPageText=" 首 页 "  LastPageText=" 尾 页 "  Mode="Next
PreviousFirstLast" NextPageText="下一页" PreviousPageText="上一页" />
```


刷新页面查看效果，如图 10-26 所示。



图 10-26 GridView 控件实现分页 2

提 示

除了前面表中列出的属性外，GridView 控件还包含许多其他的属性，如 FooterStyle、HeaderStyle、EditRowStyle 和 RowStyle 等样式属性。实际上，GridView 控件和 DataList 控件一样，可以自动套用格式，这时会自动为其指定样式。

10.6.5 GridView 控件的事件

如果用户细心，单击图 10-25 或者图 10-26 中的分页链接时会发现出现错误，提示 GridView 控件激发了未处理的 PageIndexChanging 事件。添加 GridView 控件的 PageIndexChanging 事件，代码如下。

```
protected void GridView1_PageIndexChanging(object sender, GridView
PageEventArgs e)s {
    GridView1.PageIndex = e.NewPageIndex;
    GridView1.DataSource = SqlHelper.GetDataSet(SqlHelper.connString,
    CommandType.Text, "SELECT * FROM BookInfo", null);
    GridView1.DataBind();
}
```

在上述代码中，重新指定 GridView 控件的 PageIndex 属性值，然后重新绑定 GridView 控件。重新运行上述代码查看效果，如图 10-27 所示。



图 10-27 查看最后一页

GridView 控件提供一系列的事件，除了上面提到的 PageIndexChanging 事件外，表 10-15 中还列举了其他事件。

表 10-15 GridView 控件的常用事件

事件名称	说明
PageIndexChanging	在单击某一页导航按钮时，但在控件处理分页操作之前发生
PageIndexChanged	在单击某一页导航按钮时，但在控件处理分页操作之后发生
RowCommand	当单击 GridView 控件中的按钮时发生
RowDataBound	在 GridView 控件中将数据行绑定到数据时发生
RowsCreated	在 GridView 控件中创建行时发生
RowDeleted	在单击某一行的【删除】按钮时，但在 GridView 控件删除该行之后发生
RowDeleting	在单击某一行的【删除】按钮时，但在 GridView 控件删除该行之前发生
RowEditing	发生在单击某一行的【编辑】按钮以后，GridView 控件进入编辑模式之前
RowUpdated	发生在单击某一行的【更新】按钮，并且 GridView 控件对该行进行更新之后
RowUpdating	发生在单击某一行的【更新】按钮以后，GridView 控件对该行进行更新之前
SelectedIndexChange	发生在单击某一行的【选择】按钮，GridView 控件对相应的选择操作进行处理
SelectedIndexChanging	发生在单击某一行的【选择】按钮之后，GridView 控件对相应的选择操作进行处理之前
DataBinding	当服务器控件绑定到数据源时发生
DataBound	在服务器控件绑定到数据源后发生
Sorted	单击用于列排序的超链接时，但在此控件对相应的排序操作进行处理之后发生
Sorting	单击用于列排序的超链接时，但在此控件对相应的排序操作进行处理之前发生

在表 10-15 中，PageIndexChanging 和 PageIndexChanged 与分页有关，RowDeleted 和 RowDeleting 与删除有关，Sorted 和 Sorting 与排序有关。下面只介绍 RowDataBound 事件的使用。

在 GridView 控件中将数据行绑定到数据时会引发 RowDataBound 事件。在该事件的代码中，可以设置鼠标悬浮时的背景色，可以为按钮添加相关事件，如 onmouseover、onmouseout 和 onclick 等，还可以添加自动编号。

【范例 15】

在前面范例的基础上为 GridView 控件添加 RowDataBound 事件，在该事件代码中判断当前行是否为数据行，如果是则通过 e.Row.Attributes.Add()方法添加 onmouseover 和 onmouseout 事件，指定鼠标悬浮和离开时的背景颜色。代码如下。

```
protected void GridView1 RowDataBound(object sender, GridViewRow
EventArgs e) {
    if (e.Row.RowType == DataControlRowType.DataRow) {//当前行如果是数据行
        e.Row.Attributes.Add("onmouseover", "currentcolor=this.style.
        backgroundColor;this.style.backgroundColor='palegreen'");
        e.Row.Attributes.Add("onmouseout", "this.style.backgroundColor=
        currentcolor");
    }
}
```


重新运行页面查看效果，如图 10-28 所示。



图 10-28 鼠标悬浮时的效果

【范例 16】

在介绍 Repeater 控件的事件时，提到过生成自动编号的方法。在 GridView 控件的 RowDataBound 事件中也可以生成自动编号。步骤如下。

（1）在 GridView 控件中添加 TemplateField 字段，为每一项添加一个 Label 控件。代码如下。

```
<asp:TemplateField>
    <ItemTemplate><asp:Label ID="lblID" runat="server" Text="1"></asp:Label>
   </ItemTemplate>
</asp:TemplateField>
```

（2）在后台 RowDataBound 事件中添加代码，判断当前行是否为数据行，如果是则获取 GridView 控件中的 Label 控件，并指定该控件的 Text 属性，这里需要考虑其分页。代码如下。

```
Label lb = e.Row.FindControl("lblID") as Label;
int number = e.Row.RowIndex + 1 + (GridView1.PageIndex) * GridView1.
PageSize;
lb.Text = number.ToString();
```

（3）运行页面查看效果，如图 10-29 所示。



图 10-29 添加自动编号

10.7 实验指导——GridView 控件查看和删除数据

本章着重介绍三个数据绑定控件：Repeater 控件、DataList 控件和 GridView 控件。其中 GridView 控件的功能最为强大。本节实验指导会在前面范例的基础上实现新的功能，完成数据的查看和删除。

10.7.1 查看数据

顾名思义，查看数据是指查看数据的详细信息。ASP.NET 中提供了专门查看数据的 DetailsView 控件和 FormView 控件。

DetailsView 控件一次呈现一条表格形式的记录，并提供翻阅多条记录以及插入、更新和删除记录的功能。DetailsView 控件通常用在主/从方案中，在这种方案中，主控件（如

GridView 控件）中的所选记录决定 DetailsView 控件显示的记录。

FormView 控件与 DetailsView 控件类似，它一次呈现数据源中的一条记录，并提供翻阅多条记录以及插入、更新和删除记录的功能。它与 DetailsView 控件的差别在于：DetailsView 控件使用基于表的布局，在这种布局中，数据记录的每个字段都显示为控件中的一行。而 FormView 控件则不指定用于显示记录的预定义布局。本节以 DetailsView 控件为例，演示该控件的使用。步骤如下。

（1）创建一个新的操作页面，向页面中添加 GridView 控件，自定义字段显示列，包括编号、标题、作者和操作。代码如下。

```
<asp:GridView ID="GridView1" runat="server" Width="85%" AutoGenerate
Columns="False" AllowPaging="true" PageSize="5" OnPageIndexChanging
="GridView1 PageIndexChanging" OnRowDataBound="GridView1 RowDataBound"
OnRowCommand="GridView1 RowCommand">
    <Columns>
        <!-- 省略代码 -->
        <asp:TemplateField HeaderText="操作">
            <ItemTemplate>
                <asp:LinkButton ID="lbDetails" runat="server" CommandName=
                "Details" CommandArgument='<%#Eval("BookIsbn") %>' Text="
                详细"></asp:LinkButton>   
                <asp:LinkButton ID="lbDelete" runat="server" CommandName
                ="Delete" CommandArgument='<%#Eval("BookIsbn") %>' Text="
                删除"></asp:LinkButton>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
    <PagerSettings FirstPageText="首页" LastPageText="尾页" Mode="Next
    PreviousFirstLast" NextPageText="下一页" PreviousPageText="上一页" />
</asp:GridView>
```

（2）根据步骤（1）中 GridView 控件的内容为其添加事件，这里只显示 GridView 控件 RowCommand 事件的代码。内容如下。

```
protected void GridView1 RowCommand(object sender, GridViewCommand
EventArgs e) {
    string name = e.CommandName;                    //获取命令
    string value = e.CommandArgument.ToString();    //获取值
    if (name == "Details") {
        Response.Redirect("GridViewDetails.aspx?isbn=" + value);
    } else if (name == "Delete") {
        //省略删除代码
    }
}
```

在上述代码中，首先获取 CommandNmae 和 CommandArgument 属性的值，然后根据 name 变量的值进行判断。

（3）运行页面查看效果，如图 10-30 所示。



图 10-30　页面初始效果

（4）创建 GridViewDetails.aspx 页面，将 DetailsView 控件拖入到表单元素内。在【设计】窗口中选中 GridView 控件打开任务菜单，如图 10-31 所示。

（5）单击图 10-31 中的【新建数据源】项，弹出如图 10-32 所示的对话框。



图 10-31　DetailsView 控件的任务

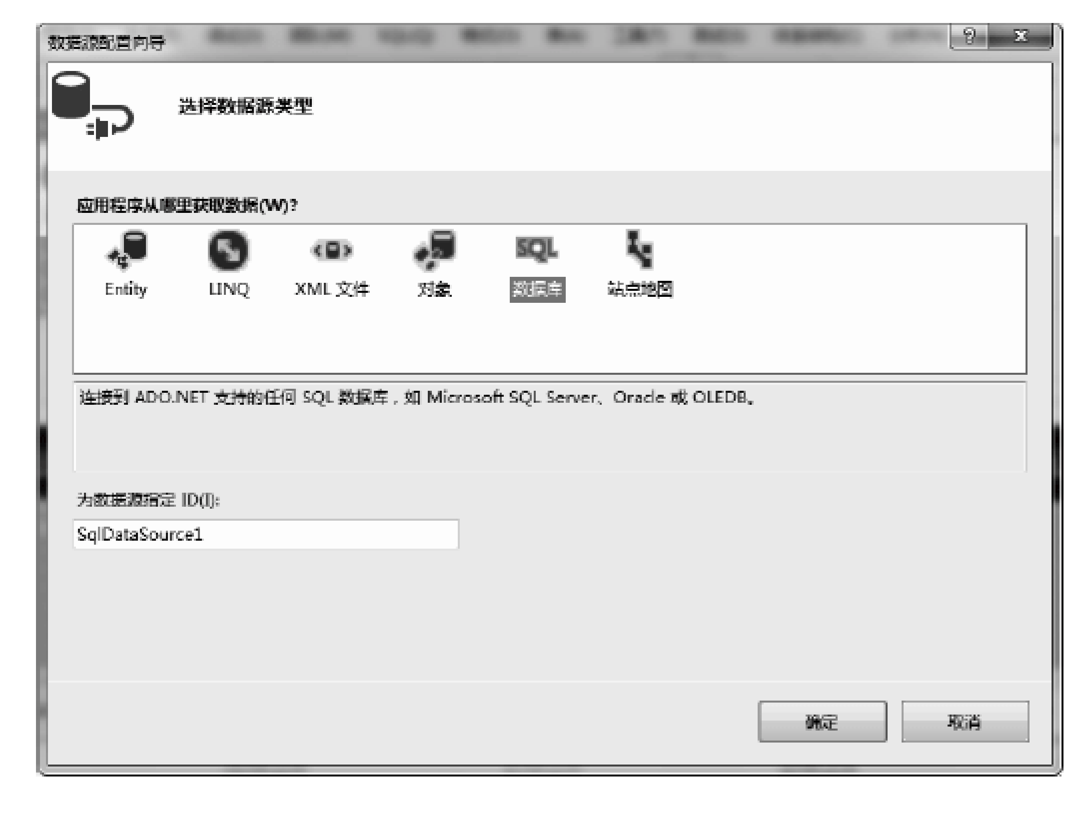


图 10-32　选择数据源类型

（6）在图 10-31 中选择数据源后单击【确定】按钮，如图 10-33 所示。

（7）在图 10-33 中选择已经存在的数据库连接，这时【上一步】和【下一步】等按钮都可用，单击【下一步】按钮，弹出【配置 Select 语句】对话框。在该对话框中单击 WHERE 按钮，弹出如图 10-34 所示对话框。

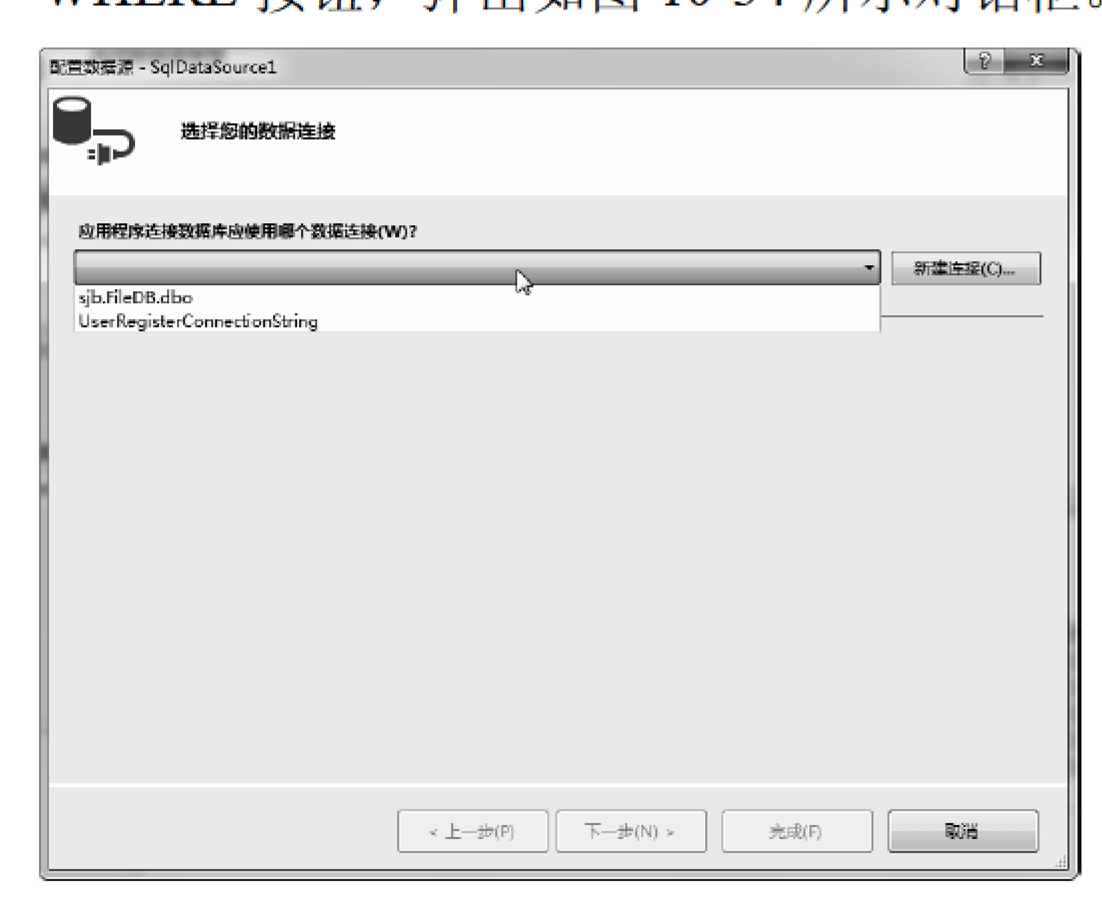


图 10-33　选择数据连接

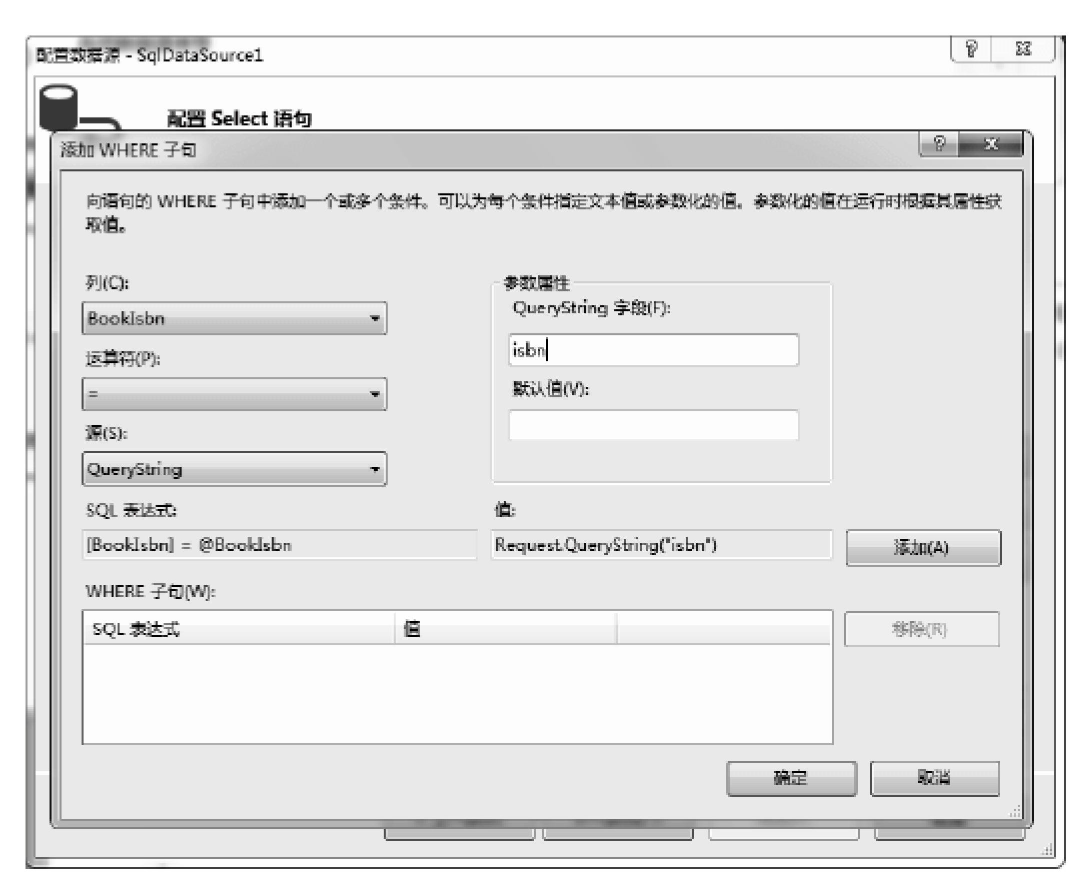


图 10-34　添加 WHERE 子名

（8）在图 10-34 中选择列和源，并输入参数属性，然后单击【添加】按钮将其添加到 WHERE 子句中，完毕后单击【确定】按钮。

（9）根据提示执行其他的操作，效果图不再显示。所有的操作完毕后，会在 GridViewDetails.aspx 页面中自动添加代码，具体代码不再显示。

（10）运行页面进行测试，单击图 10-30 中某一条记录后的【详细】链接查看详细信息，如图 10-35 所示。

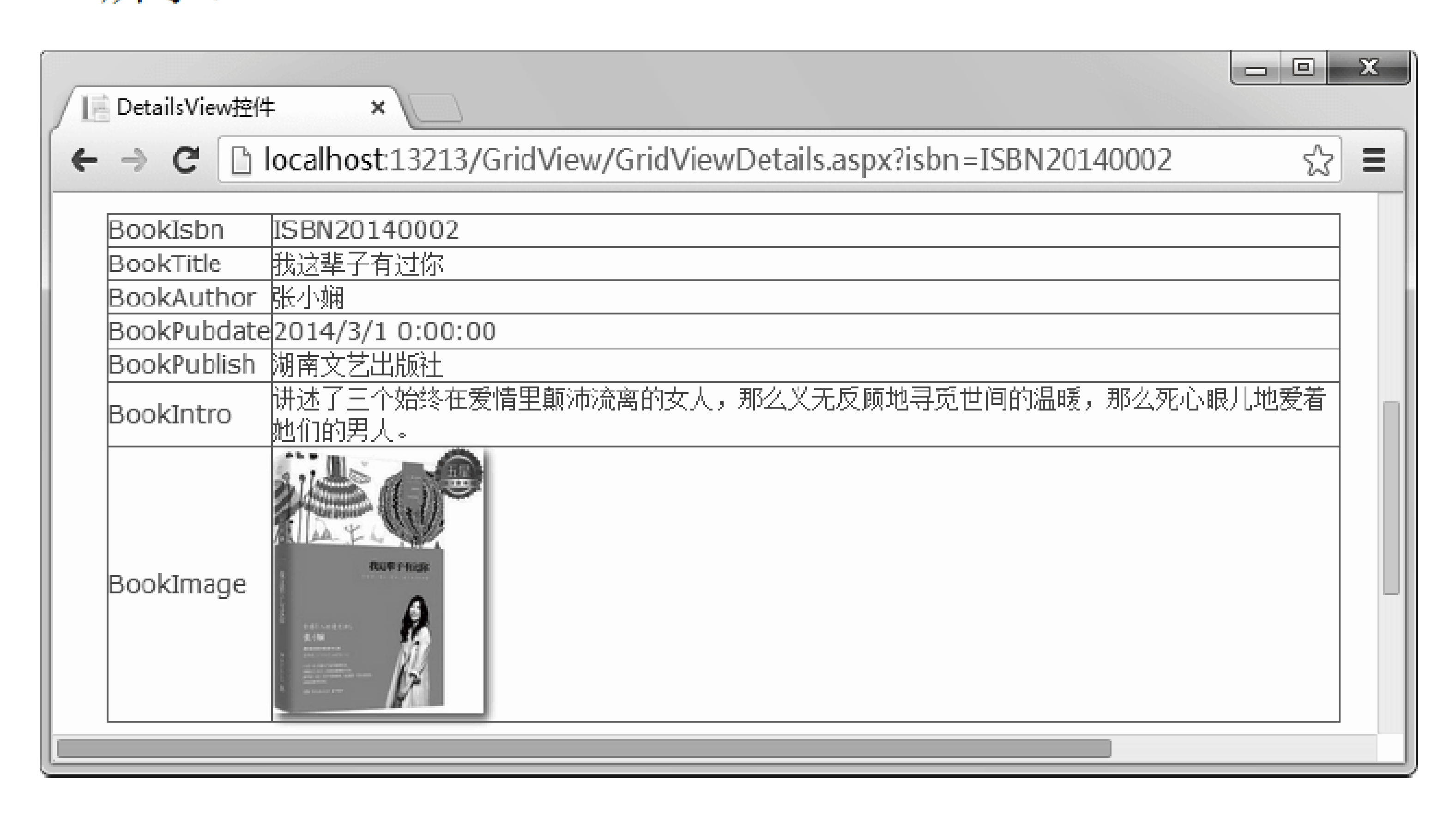


图 10-35 查看详细信息

> **提示**
> 读者可以直接通过上述方法绑定数据，也可以手动编写代码实现绑定。手动编写代码时，需要在页面后台接收从上个页面传递过来的参数，根据该参数获取对象，具体代码不再显示。

10.7.2 删除数据

单击图 10-30 中的【删除】链接可以删除某一条记录。代码如下。

```
int result = SqlHelper.ExecuteNonQuery(SqlHelper.connString, CommandType.
Text, "DELETE FROM BookInfo WHERE BookIsbn='" + value + "'", null);
    if (result > 0) {
        Page.ClientScript.RegisterStartupScript(this.GetType(),"", "<script>
        alert('删除成功');</script>");
        //重新绑定 GridView 控件
    } else {
        Page.ClientScript.RegisterStartupScript(this.GetType(),"","<script>
        alert('删除失败');</script>");
}
```

如果有需要，还要为 GridView 控件添加 RowDeleting 事件。执行上述代码查看效果，效果图不再显示。

提示

本章着重介绍 Repeater、DataList 和 GridView 控件，并且演示了 DetailsView 控件的使用。除了这些控件外，ASP.NET 还提供了其他的数据绑定控件，这里不再一一介绍。

思考与练习

一、填空题

1．在进行数据绑定时，__________方法用于单向绑定。

2．数据源控件__________专门处理类似站点地图的 XML 数据。

3．DataList 控件的__________属性用于获取或设置要在控件中显示的列数。

4．将 GridView 控件__________属性的值设置为 true，表示启动分页功能。

二、选择题

1．Repeater 控件的模板不包括__________。

A．HeaderTemplate

B．ItemTemplate

C．SeparatorTemplate

D．SelectedItemTemplate

2．PagedDataSource 类实现分页功能时，DataSourceCount 属性用于__________。

A．获取数据源中的项数

B．获取或设置数据源

C．获取要从数据源使用的基数

D．获取显示数据源中的所有项所需要的总页数

3．关于 Repeater、DataList 和 GridView 控件，下面说法正确的是__________。

A．Repeater 控件和 GridView 控件内置分页和排序功能

B．只有 GridView 控件内置分页和排序功能

C．只有 Repeater 控件可以通过 PagedDataSource 类实现分页功能，DataList 和 GridView 控件不能使用 PagedDataSource 类

D．GridView 控件的功能最为强大和复杂，DataList 控件的功能最简单

4．GridView 控件的__________属性用于获取或设置控件在每页上所显示的记录条数。

A．PageCount

B．PageIndex

C．PageSize

D．PageSettings

5．为 GridView 控件添加 RowDataBound 事件代码，首先判断当前行是否为数据行，横线处应该填写__________。

```
protected void gvShow RowDataBound
(object sender, GridViewRowEvent
Args e) {
    if (e.Row.RowType ==       ){
        /* 省略代码 */
    }
}
```

A．DataControlCellType.DataRow

B．DataControlRowType.DataRow

C．DataControlRowType.Pager

D．DataControlRowType.EmptyDataRow

三、简答题

1．常见的数据绑定有哪几种？

2．ASP.NET 中提供的数据源控件有哪些？这些控件分别用来做什么？

3．Repeater 控件常用的属性和事件有哪些？它们分别用来做什么？

4．DataList 控件的常用属性和事件有哪些？它们分别用来做什么？

5．GridView 控件的常用属性和事件有哪些？它们分别用来做什么？

第 11 章　LINQ 数据处理

LINQ 在对象领域和数据领域之间架起了一座桥梁。它将数据查询操作直接引入到C#中，从而直接实现查询、更新和删除功能，而不是以字符串嵌入到应用程序代码中。使用 LINQ 可以大量减少查询或操作数据库或数据源中的数据的代码，并在一定程度上避免了 SQL 注入，提高了应用程序的安全性。

本章主要介绍 LINQ 的组成部分、各子句的应用以及 LINQ to SQL 操作数据库的方法。

本章学习要点：

- ❑ 了解 LINQ 语句的组成
- ❑ 掌握各个 LINQ 子句的使用
- ❑ 熟悉 LINQ 连接多个数据源的方法
- ❑ 掌握对象关系设计器的创建
- ❑ 掌握数据的插入、更新和删除

11.1　LINQ 概述

查询通常用专门的查询语言来表示，例如关系数据库中的 SQL 和 XML 中的 XQuery。LINQ 将查询作为 C#语言的一种内置特性，提供一种跨各种数据源和数据格式使用数据的一致模型，使开发人员可以使用相同的编码模式来查询和转换 XML 文档、SQL 数据库、ADO.NET 数据集、.NET 集合中的数据以及 LINQ 提供程序的任何其他格式的数据。

11.1.1　LINQ 类型

LINQ 是 Language Integrated Query（语言集成查询）的简称，可以查询数据源包含一组数据的集合对象（IEnumerable<T>或者 IQueryable<T>类），返回的查询结果也是一个包含一组数据的集合对象。所以，编译时将对查询的数据类型进行检查，增强了类型安全性。同时使用 VS 2012 提供的智能提示，使得编码更加快捷。LINQ 还可以通过函数的形式提供过滤条件等，大大简化了查询的复杂度。

由于 LINQ 中查询表达式访问的是一个对象，所以该对象可以表示各种类型的数据源。在.NET Framework 类库中，与 LINQ 相关的类都在 System.Linq 命名空间下。该命名空间提供支持使用 LINQ 进行查询的类和接口，其中最常用的有如下类和接口。

（1）IEnumerable<T>接口：它表示可以查询的数据集合，一个查询通常是逐个对集合中的元素进行筛选操作，返回一个新的 IEnumerable<T>对象用来保存查询结果。

（2）Enumerable 类：它通过对 IEnumerable<T>提供扩展方法，实现 LINQ 标准查询

运算符，包括过滤、排序、查询、联接、求和等操作。

（3）IQueryable<T>接口：它继承 IEnumerable<T>接口，表示一个可以查询的表达式目录树。

（4）Querable 类：它通过对 IQueryable<T>提供扩展方法，实现 LINQ 标准查询运算符，包括过滤、排序、查询、联接、求和等操作。

> 提示
>
> *在学习 LINQ 之前，读者应该熟悉使用 LINQ 所需的 C#高级特性，如接口、泛型、扩展方法和匿名类型等。*

根据数据源类型的不同，LINQ 技术可以分为如下 4 种类型。

（1）LINQ to Objects：查询任何可枚举的集合，如数组、泛型列表和字典等。

（2）LINQ to SQL：查询和处理基于关系数据库的数据。

（3）LINQ to DataSet：查询和处理 DataSet 对象中的数据，并对这些数据进行检索、过滤和排序等操作。

（4）LINQ to XML：查询和处理基于 XML 结构的数据。

如图 11-1 所示描述了 LINQ 如何关联到其他数据源和高级编程语言。

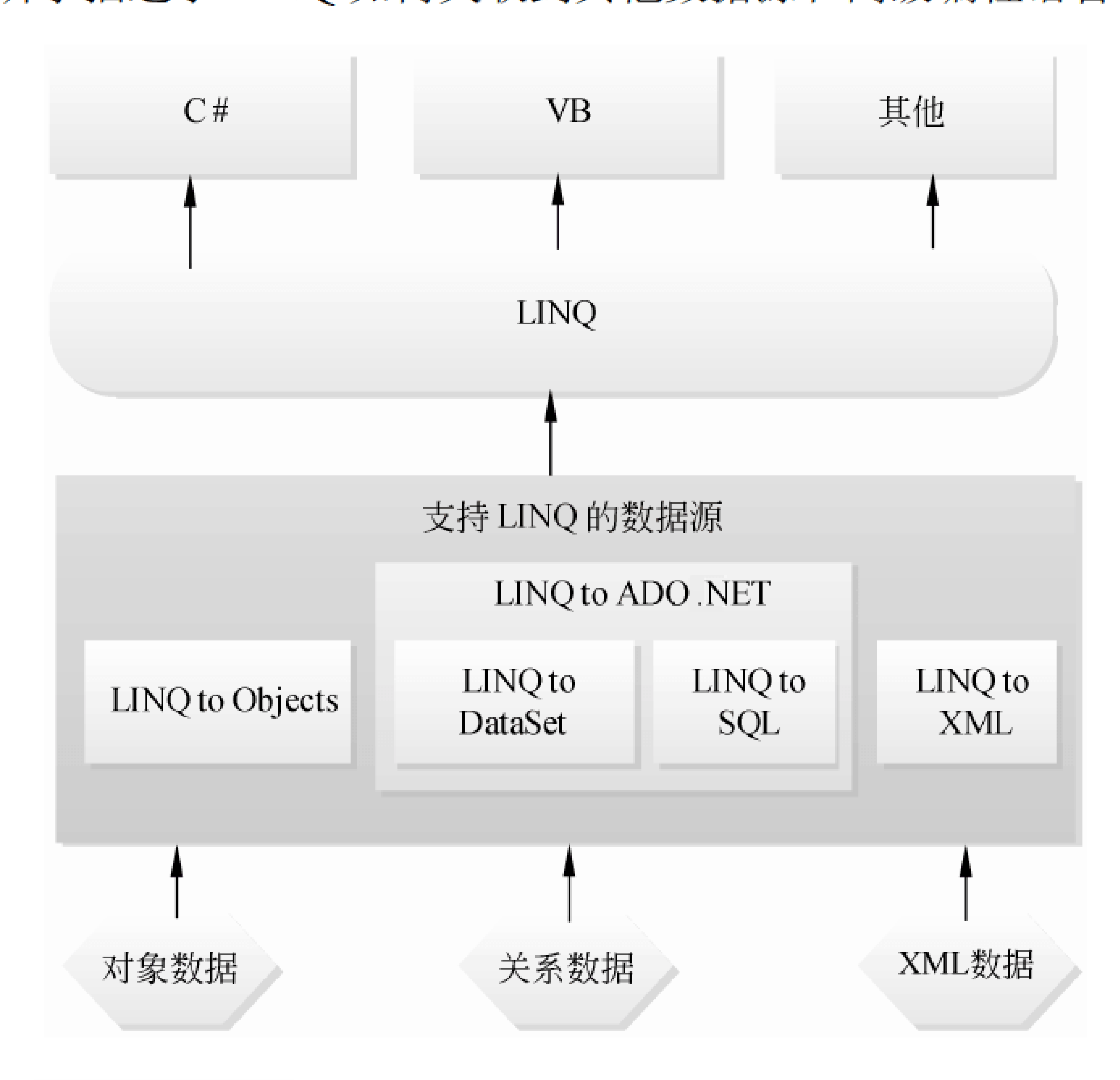


图 11-1　LINQ 相关技术描述图

【范例 1】

要使用 LINQ，首先必须引用与 LINQ 相关的命名空间。例如，要使用 LINQ to XML 和 LINQ to SQL 需要如下的命名空间。

```
using System.Linq;
using System.Xml.Linq;
```

```
using System.Data.Linq;
```

除了 System.Linq 命名空间会自动导入之外，其他两个需要用户手动添加。

11.1.2 LINQ 查询语句解析

引用 LINQ 命名空间之后，在 C#中可以非常简单地使用 LINQ 查询。只需将它看作普通的对象编写代码即可。例如，如下的 SQL 语句从 students 数据表中查询出学生姓名。在该语句中使用查询关键字来表示特定的功能，包括指定数据源、查询结果、筛选条件等。

```
select name from students
```

在上述语句中 select 和 from 是关键字，分别用来指定要查询的结果和数据源。

LINQ 中的查询与传统查询类似，同样可以采用具有一定语义的文本来表示。例如，如下是从 students 数据表中查询出学生姓名的 LINQ 实现。

```
var result = from num in numbers select num;
```


这种方式在 LINQ 中称为查询表达式。在这里的 from 和 select 都是 LINQ 中的子句，11.2 节将详细介绍这些子句及它们的具体作用。

提 示

LINQ 中的查询同时是一个类型为 IEnumerable<T>或者 IQuerable<T>的对象，所以可以通过使用对象的方式使用它，这种方式在 LINQ 中称为查询方法。

LINQ 查询的目的是从指定的数据源中查询满足特定条件的数据元素，并且根据需要对这些元素进行排序、分组、统计及联接等操作。所以，一个 LINQ 查询应该包含如下几个主要元素。

1. 数据源

数据源表示 LINQ 查询将从哪里获取数据，它通常是一个或者多个数据集，每一个数据集包含一系列的元素。数据集是一个类型为 IEnumerable<T>或者 IQueryable<T>的对象，可以对它进行枚举，遍历每一个元素。此外，它的元素可以是任意数据类型，所以可以表示任何数据的集合。

2. 目标数据

数据源中的元素并不是查询所需要的结果。例如，对于一个学生信息集合，查询 1 只需要查询学生学号，查询 2 只需要查询学生姓名和班级编号，查询 3 则需要学生学号和入学时间。因此，目标数据用来指定查询具体想要的数据，在 LINQ 中它定义了查询结果数据集中元素的具体类型。

3．筛选条件

筛选条件定义了对数据源中元素的过滤条件。只有满足条件的元素才作为查询结果返回。筛选条件可以是简单的逻辑表达式，也可以是具有复杂运算的函数。

4．附加操作

附加操作表示一些其他的具体操作。例如，对查询的结果进行排序、计算查询结果中的最大值，或者进行分组等。

其中，每个 LINQ 查询必须具有数据源和目标数据两个元素，筛选条件和附加操作是可选元素。

11.2 LINQ to Object

凡是使用 LINQ 查询表达式查询的数据，都属于 LINQ to Object。LINQ to Object 是指直接对任意 IEnumerable 或 IEnumerable<(Of <(T>)>)集合使用 LINQ 查询，无须使用中间 LINQ 提供程序或 API，如 LINQ to SQL 或 LINQ to XML。

LINQ to Object 可以使用查询任何可枚举的集合，如 List<(Of <(T>)>)、Array 或 Dictionary<(Of <(TKey, TValue>)>)等。该集合可以是用户定义的集合，也可以是.NET Framework API 返回的集合。从根本上说，LINQ to Objects 表示一种新的处理集合的方法。采用旧方法，必须编写指定如何从集合检索数据的复杂的 foreach 循环。而如果采用 LINQ 方法，程序员只需编写描述要检索的内容的声明性代码就可以得到要检索的数据。

11.2.1 了解 LINQ 子句

查询表达式是由查询关键字和对应的操作数组成的表达式整体。其中，查询关键字包括与 SQL 对应的子句，这些子句的介绍如表 11-1 所示。

表 11-1 查询关键字和子句

关键字	说明
from	指定要查找的数据源以及范围变量，多个 from 子句则表示从多个数据源查找数据
where	指定元素的筛选条件，多个 where 子句则表示了并列条件，必须全部都满足才能入选
select	指定查询要返回的目标数据，可以指定任何类型，甚至是匿名类型
group	按照指定的键值对查询结果进行分组
into	提供一个标识符，它可以充当对 join、group 或 select 子句的结果的引用
orderby	基于元素类型的默认比较器按升序或降序对查询结果进行排序
join	基于两个指定匹配条件之间的相等比较来联接两个数据源
let	引入一个用于存储查询表达式中的子表达式结果的范围变量

LINQ 查询表达式必须以 from 子句开头，并且必须以 select 或 group 子句结尾。在第一个 from 子句和最后一个 select 或 group 子句之间，查询表达式可以包含一个或多个下列可选子句：where、orderby、join、let 甚至附加的 from 子句。还可以使用 into 关键

字使 join 或 group 子句的结果能够充当同一查询表达式中附加查询子句的源。

11.2.2 FROM 子句

在一个查询中数据源是必不可少的元素。LINQ 的数据源是实现泛型接口 IEnumerable<T>或者 IQueryable<T>的类对象，可以使用 foreach 遍历它的所有元素，从而完成查询操作。

通过为泛型数据源指定不同的元素类型，可以表示任何数据集合。C#中的列表类、集合类以及数组等都实现了 IEnumerable<T>接口，所以可以直接将这些数据对象作为数据源在 LINQ 查询中使用。

LINQ 使用 from 子句来指定数据源，它的语法格式如下。

```
from 变量 in 数据源
```

一般情况下，不需要为 from 子句的变量指定数据类型，因为编译器会根据数据源类型自动分配，通常元素类型为 IEnumerable<T>中的类型 T。例如，当数据源为 IEnumerable<string>时，编译器为变量分配 string 类型。

【范例 2】

创建一个 string 类型的数组，然后将它作为数据源使用 LINQ 查询所有元素。示例代码如下。

```
string[] Colors = { "Black", "Red", "Yello", "Green" };
                                            //定义 string 类型数组 Colors
var result = from color in Colors
           select color;                    //LINQ 查询所有元素
```

在上述语句中由于 Colors 数组是 string 类型，所以在 LINQ 查询时的 color 变量也是 string 类型。

【范例 3】

如果不希望 from 子句中的变量使用默认类型，也可以指定数据类型。例如，下面的代码将 from 子句中的变量转换为 object 类型。

```
string[] Colors = { "Black", "Red", "Yello", "Green" };
                                            //定义 string 类型数组 Colors
var result = from object color in Colors
           select color;                    //LINQ 查询所有元素
```

【范例 4】

在使用 from 子句时要注意，由于编译器不会检查遍历变量的类型。所以当指定的数据类型无法与数据源的类型兼容时，虽然不会有语法上的错误，但是当使用 foreach 语句遍历结果集时将会产生类型转换异常。

示例代码如下。

```
string[] enOfNum = { "one", "two", "three", "four" };
```

```
                                                  //定义 string 类型数组 enOfNum
    var result = from int num in enOfNum
           select num;                            //使用 int 类型变量查询 enOfNum 数据源
     foreach (var str in result)                  //遍历查询结果的集合
     {
        Response.Write(str);                      //输出集合中的元素
     }
```

上述代码创建一个 string 类型的数据源，然后使用 int 类型的变量来进行查询，由于 int 类型和 string 类型不兼容，也不能转换，所以在运行时将会产生异常信息，如图 11-2 所示。



图 11-2　异常信息

注意

建议读者不要在 from 子句中指定数据类型，应该让编译器自动根据数据源分配数据类型，避免发生异常信息。

11.2.3　SELECT 子句

select 子句用于指定执行查询时产生结果的结果集，它也是 LINQ 查询的必备子句。语法如下。

```
select 结果元素
```

其中，select 是关键字，结果元素则指定了查询结果集合中元素的类型及初始化方式。如果不指定结果的数据类型，编译器会将查询中结果元素的类型自动设置为 select 子句中元素的类型。

【范例 5】

例如，本范例从 int 类型的数组中执行查询，而且在 select 子句中没有指定数据类型。此时 num 结果元素将会自动编译为 int 类型，因为 result 的实际类型为 IEnumerable<int>。

```
int[] numbers = { 1, 2, 3, 4};                //创建数据源
var result = from num in numbers
        select num;                           //查询数据
Console.SetOut(Response.Output);
foreach (var num in result)                   //遍历结果集合 result
 {
    Response.Write(num + "|");                //每个集合元素为 int 类型
 }
```

执行后的查询结果如下。

```
1|2|3|4|
```

【范例 6】

在范例 5 中使用 select 子句获取数据源的所有元素。该子句还可以获取目标数据源中子元素的操作结果，例如属性、方法或者运算等。

例如，本范例从一个包含商品名称、价格和数量的数据源获取商品名称信息。

```
var Foods = new[]                             //定义数据源
 {
    new{name="面包",price=5,amount=25},
    new{name="饼干",price=6,amount=21},
    new{name="饮料",price=3,amount=22},
    new{name="瓜子",price=4,amount=24}
 };                                           //通过匿名类创建商品列表对象
var foodNames = from foods in Foods
             select foods.name;               //从数据源中查询 name 属性
Console.SetOut(Response.Output);
foreach (var str in foodNames)                //遍历结果
{
   Console.WriteLine("名称：" + str);
}
```

上述代码首先创建一个名为 Foods 的数据源，在该数据源中通过 new 运算符创建 4 个匿名的商品信息对象。每个对象包含三个属性：name（商品名称）、price（价格）和 amount（数量）。

LINQ 语句从数据源中查询所有对象，每个对象为一个匿名商品信息，select 子句从商品信息中提取 name 属性，即名称信息。运行之后的输出结果如下。

```
名称：面包
名称：饼干
名称：饮料
名称：瓜子
```

【范例 7】

如果希望获取数据源中的多个属性，可以在 select 子句中使用匿名类型来解决。以范例 6 中的数据源为例，现在要查询出商品名称和价格信息，实现代码如下。

```
var stus = from stu in students
        select new { stu.name, stu.age };
 foreach (var stu in stus)
 {
    Response.Write("名称：{0}   价格:{1}<br/>", stu.name, stu.age);
 }
```

上述语句在 select 子句中使用 new 创建一个匿名类型来描述包含学生姓名和年龄的对象。由于在 foreach 遍历时无法表示匿名类型，所以只能通过 var（可变类型）关键字让编译器自动判断查询中元素的类型。运行之后的输出结果如下。

```
名称：面包   价格:5
名称：饼干   价格:6
名称：饮料   价格:3
名称：瓜子   价格:4
```

> **技巧** 通常情况下，不需要为 select 子句的元素指定具体数据类型。另外，如果查询结果中的元素只在本语句内临时使用，应该尽量使用匿名类型，这样可以减少很多不必要类的定义。

11.2.4　WHERE 子句

在实际查询时并不总是需要查询出所有数据，而是希望对查询结果的元素进行筛选。只有符合条件的元素才能最终显示。在 LINQ 中使用 where 子句来指定查询的条件，语法格式如下。

```
where 条件表达式
```

这里的条件表达式可以是任何符合 C#规则的逻辑表达式，最终返回 true 或者 false。当被查询的元素与条件表达式的运算结果为 true 时，该元素将出现在结果中。

【范例 8】

假设要从一个 int 类型数据源中查询出数字大于 4 的元素。实现代码如下。

```
int[] numbers ={ 1, 8, 2, 5,6,3,3,7,10 };  //创建数据源
 var result = from num in numbers
              where num>4
              select num;                    //使用 where 子句查询大于 4 的元素
 foreach (var num in result)                 //遍历查询结果
 {
    Response.Write (num + "|");
 }
```

上述代码使用 where 子句对 num 进行筛选，只有满足 num>4 条件的元素会出现在结果集合中。运行结果如下。

```
8|5|6|7|10|
```

下面的语句从数据源中查询出大于 4 的偶数。

```
var result = from num in numbers
        where (num%2==0)&&(num>4)
        select num;
```

这里使用了与运算符&&连接两个表达式。

【范例 9】

从范例 6 创建的商品信息数据源中执行如下条件查询。

（1）查询价格为 5 的商品名称和数量。

```
var result = from food in Foods
       where f.price==5
       select new { food.name, food.amount };
```

（2）查询价格大于 3 的商品名称。


```
var result = from food in Foods
       where food.price>3
       select food.name;
```

（3）查询价格大于 4，并且数量大于 10 的商品名称。

```
var result = from food in Foods
       where (food.price>4) && (food.sex>10)
       select food.name;
```

（4）查询价格等于 4，或者数量等于 10 的商品名称、价格和数量。

```
var result = from food in Foods
       where (food.price>4) || (food.sex>10)
       select new {food.name, food.price , food.amount};
```

11.2.5 ORDERBY 子句

LINQ 查询使用 orderby 子句来对结果中的元素进行排序，语法格式如下。

```
orderby 排序元素 [ascending | descending]
```

其中，排序元素必须在数据源中，中括号内是排序方式，默认是 ascending 关键字表示升序排列，descending 关键字表示降序排列。

【范例 10】

本范例演示使用 orderby 子句对整型数据源的升序和降序操作，并输出结果。实现代码如下。

```
int[] numbers = { 1, 8, 2, 5, 6, 3, 3, 7, 10 };    //定义数据源
var result1 = from num in numbers
          orderby num
          select num;                              //使用默认升序排列
Response.Write("默认排序结果是：<br>");
foreach (var num in result1){
   Response.Write(num + " ,");
}
var result2 = from num in numbers
          orderby num descending
          select num;                              //指定降序排列
Response.Write("<br>降序排列结果是：");
foreach (var num in result2){
   Response.Write(num + " ,");
}
```

输出结果如下。

```
默认排序结果是：
1 ,2 ,3 ,3 ,5 ,6 ,7 ,8 ,10 ,
降序排列结果是：
10 ,8 ,7 ,6 ,5 ,3 ,3 ,2 ,1 ,
```

【范例 11】

对商品信息按价格降序排列，输出商品名称和价格。代码如下。

```
var foods = from food in Foods
         orderby food.price descending
         select new { food.name, food.price };
 foreach (var f in foods)
 {
     Response.Write(string.Format("名称：{0}价格:{1}<br>",f.name,f.price));
 }
```

运行后的输出结果如下。

```
名称：饼干 价格:6
名称：面包 价格:5
名称：瓜子 价格:4
名称：饮料 价格:3
```

11.2.6 GROUP 子句

group 子句用于对 LINQ 查询结果中的元素进行分组，这一点与 SQL 中的 group by 子句作用相同。group 子句的语法格式如下。

```
group 结果元素 by 要分组的元素
```

其中，要分组的元素必须在结果中，而且 group 子句返回的是一个分组后的集合，集合的键名为分组的名称，集合中的子元素为该分组下的数据。因此，必须使用嵌套

foreach 循环来遍历分组后的数据。

【范例 12】

对商品信息列表中的单位进行分组，输出每组下的商品名称和价格。实现代码如下。

```
var Foods = new[]                              //定义数据源
 {
     new{name="面包",price=5,amount=25,quality="袋"},
     new{name="饼干",price=6,amount=21,quality="袋"},
     new{name="饮料",price=3,amount=22,quality="瓶"},
     new{name="瓜子",price=4,amount=24,quality="袋"},
     new{name="啤酒",price=5,amount=22,quality="瓶"}
 };
var foods = from food in Foods
          group food by food.quality;  //对 Foods 数据源按照 quality 元素进行分组
foreach (var group in foods)           //遍历分组结果
{
    Response.Write(String.Format("单位为{0}的共{1}个<br>", group.Key,
    group.Count()));
    foreach (var f in group)
    {
        Response.Write(string.Format("名称：{0}   价格：{1}<br>", f.name,
         f.price));
    }
}
```


上述代码将分组结果保存在 foods 变量中，此时该变量是一个二维数组。在遍历二维数组时通过调用 Key 属性获取分组的名称，调用 Count()方法获取该组子元素的数量，最后再使用一个 foreach 循环遍历子元素的内容。

运行后输出结果如下。

```
单位为袋的共 3 个
名称：面包  价格：5
名称：饼干  价格：6
名称：瓜子  价格：4
单位为瓶的共 2 个
名称：饮料  价格：3
名称：啤酒  价格：5
```

11.2.7 JOIN 子句

与 SQL 一样，LINQ 也允许根据一个或者多个关联键来联接多个数据源。join 子句实现了将不同的数据源在查询中进行关联的功能。

join 子句使用特殊的 equals 关键字比较指定的键是否相等，并且 join 子句执行的所有联接都是同等联接。join 子句的输出形式取决于所执行联接的类型，联接类型包括：内联接、分组联接和左外联接。

本节用到两个数据源，第一个是图书分类信息如表 11-2 所示，第二个是图书详细信息如表 11-3 所示。

表 11-2 图书分类信息表

分类编号	分类名称	上级分类编号
1	软件开发	0
2	数据库	0
3	办公软件	0
4	C#程序设计	1
5	Java 程序设计	1
6	SQL Server 数据库	2

表 11-3 图书详细信息表

图书编号	图书名称	图书价格	分类编号
1	C#入门与提高	105	4
2	C#程序设计标准教程	98	4
3	轻松学 Java 编程	58	5
4	轻松学 C#编程	59	4
5	轻松学 Word 软件	62	3
6	SQL 入门很简单	55	6

1. 内联接

内联接是 LINQ 查询中最简单的一种联接方式。它与 SQL 语句中的 INNER JOIN 子句比较相似，要求元素的联接关系必须同时满足被联接的两个数据源，即两个数据源都必须存在满足联接关系的元素。

【范例 13】

使用内联接将分类编号作为关联查询图书名称、价格和分类名称。实现步骤如下。

（1）创建保存图书分类信息的数据源，代码如下。

```
var Types = new[]
 {
    new {tid=1,tname="软件开发",pid=0},
    new {tid=2,tname="数据库",pid=0},
    new {tid=3,tname="办公软件",pid=0},
    new {tid=4,tname="C#程序设计",pid=1},
    new {tid=5,tname="Java 程序设计",pid=1},
    new {tid=6,tname="SQL Server 数据库",pid=2}
 };
```

（2）创建保存图书信息的数据源，代码如下。

```
var Books = new[]
{
   new{bkid=1,bkname="C#入门与提高",price=105,tid=4},
   new{bkid=2,bkname="C#程序设计标准教程",price=98,tid=4},
   new{bkid=3,bkname="轻松学 Java 编程",price=58,tid=5},
   new{bkid=4,bkname="轻松学 C#编程",price=59,tid=4},
   new{bkid=5,bkname="轻松学 Word 软件",price=62,tid=3},
   new{bkid=6,bkname="SQL 入门与提高",price=55,tid=6}
};
```

（3）创建 LINQ 查询，使用 join 子句按 tid 元素关联分类信息和图书信息，并选择图书编号、价格和分类名称作为结果。实现语句如下。

```
var result = from bk in Books
        join t in Types
        on bk.tid equals t.tid
        select new
        {
            bookName = bk.bkname,
            bookPrice = bk.price,
            typeName = t.tname
        };
```

上述语句指定以 Books 数据源为依据，根据 tid 键在 Types 数据源中查找匹配的元素。

> **提 示**
>
> 与 SQL 不同，LINQ 内联接的表达式顺序非常重要。equals 关键字左侧必须是外部数据源的关联键（from 子句的数据源），右侧必须是内部数据源的关联键（join 子句的数据源）。

（4）使用 foreach 语句遍历结果集，输出每一项的内容。代码如下。

```
foreach (var item in result)
{
    body.Text+=string.Format("<tr><td>{0}</td><td>{1}</td> <td>{2}</td>
    </tr>", item.bookName, item.bookPrice, item.typeName);
}
```


（5）运行上述代码，输出结果如图 11-3 所示。



图 11-3 内联接运行效果

2. 分组联接

分组联接是指包含 into 子句的 join 子句的联接。分组联接将产生分层结构的数据，它将第一个数据源中的每个元素与第二个数据源中的一组相关元素进行匹配。第一个数据源中的元素都会出现在查询结果中。如果第一个数据源中的元素在第二个数据中找到

相关元素，则使用被找到的元素，否则使用空。

【范例 14】

使用分组联接完成如下查询。

（1）从 Types 类中查询图书分类信息。

（2）使用 join 子句联接 Types 和 Books，联接关系为相等（equal），并设置分组的标识为 g。

（3）使用 select 子句获取结果集，要求包含分类名称及其分类下的图书名称。

实现上述查询要求的 LINQ 语句如下。

```
var result = from t in Types
        join bk in Books
        on t.tid equals bk.tid into g
        select new
        {
            typeName=t.tname,
            Books=g.ToList()
        };
```

上述语句在 join 子句中通过 into 关键字将某一分类下对应的图书信息放到 g 变量中。select 子句使用 g.ToList()方法将图书信息作为集合放在结果的 Books 属性中。

使用嵌套的 foreach 遍历分组及分组中的内容，语句如下。

```
foreach (var item in result)
{
    body.Text += string.Format("分类名称[ <b>{0}</b> ]下的图书有：<ul>",
    item.typeName);
    foreach (var book in item.Books)
    {
        body.Text +=string.Format("<li>{0}</li>", book.bkname);
    }
    body.Text +="</ul>";
}
```

执行后的输出结果如图 11-4 所示。

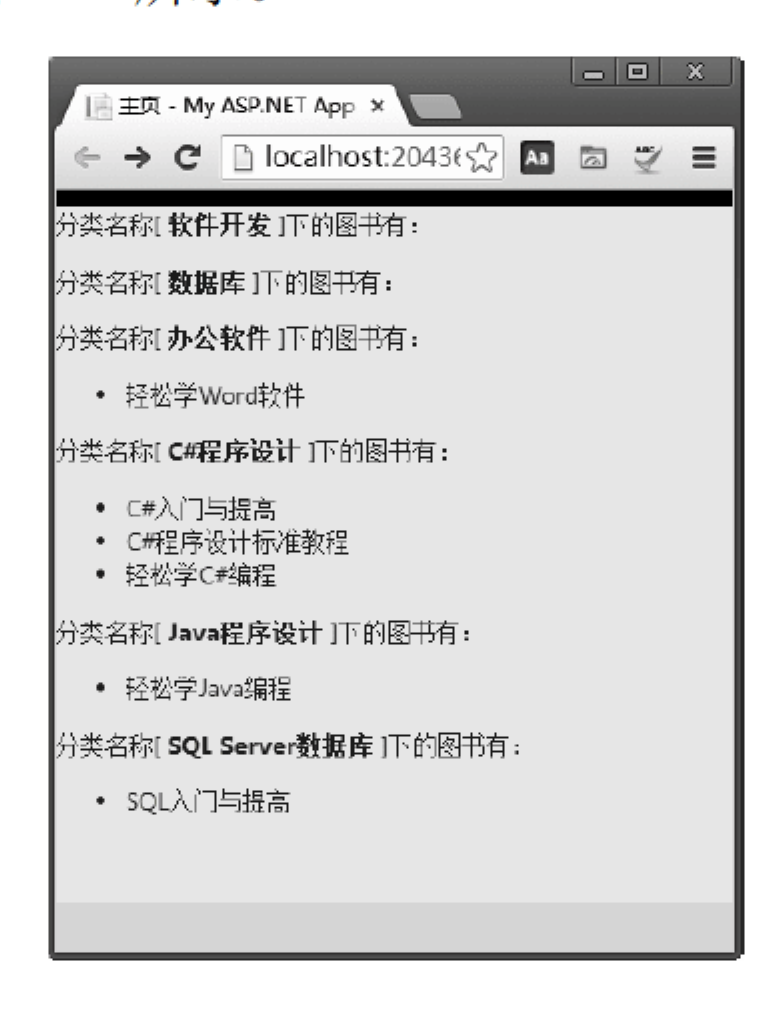


图 11-4　按分类查看图书信息

3. 左外联接

左外联接与 SQL 语句中的 LEFT JOIN 子句比较相似，它将返回第一个集合中的每一个元素，而无论该元素在第二个集合中是否具有相关元素。

LINQ 为左外联接提供了 DefaultIfEmpty()方法。如果第一个集合中的元素没有找到相关元素时，DefaultIfEmpty()方法可以指定该元素的相关元素的默认元素。

【范例 15】

使用左外联接分类信息和图书信息，并显示分类编号、分类名称和图书信息。查询语句如下。

```
var result = from t in Types
        join bk in Books
        on t.tid equals bk.tid into g
        from left in g.DefaultIfEmpty(
            new { bkid = 0, bkname = "该分类下暂无图书", price = 0, tid = t.tid }
        )
        select new
        {
            id = t.tid,
            name = t.tname,
            bookName = left.bkname
        };
```

上述使用 left 指定使用左外联接，在联接时如果找不到右侧数据源中的匹配元素则使用 DefaultIfEmpty()方法创建一个匿名类型作为值。编写 foreach 语句遍历查询结果，语句如下。

```
foreach (var item in result)
{
    body.Text+=string.Format("<tr><td>{0}</td><td>{1}</td><td> {2}</td>
    </tr>", item.id, item.name, item.bookName);
}
```

执行后的结果如图 11-5 所示。



图 11-5 左外联接运行效果

注意

左外联接和分组联接虽然相似但是并非一样。分组联接返回的查询结果是一种分层数据结构，需要使用两层 foreach 才能遍历它的结果。而左外联接是在分组联接的查询结果上再进行一次查询，所以它在 join 之后还需要一个 from 子句进行查询。

11.3 LINQ to SQL

LINQ to SQL 用于以对象形式管理关系数据，并提供了丰富的查询功能。LINQ to SQL 建立在公共语言类型系统中基于 SQL 的模式定义之上，当保持关系型模型表达能力和对底层存储的直接查询评测的性能时，这个集成在关系型数据之上提供强类型。

11.3.1 对象关系设计器简介

对象关系设计器（Object Relation 设计器，简称 OR 设计器）是 VS 2012 提供的一个可视化设计界面环境，用于创建基于数据库中对象的 LINQ to SQL 实体类和关联。OR 设计器会将数据库对象与 LINQ to SQL 类之间的映射保存到名为“.dbml”的文件中。

使用 OR 设计器可以在应用程序中创建映射到数据库对象的对象模型。同时还会生成一个 DataContext 类用于在实体类与数据库之间发送和接收数据。此外，OR 设计器还提供了将存储过程和函数映射到 DataContext 类的方法以返回数据并填充实体类的功能。OR 设计器还支持对实体类之间的继承关系进行设计。

O/R 设计器的设计界面有两个不同的区域：左侧的实体窗格以及右侧的方法窗格。实体窗格是主设计界面，其中显示实体类、关联和继承层次结构；方法窗格是显示映射到存储过程和函数的 DataContext 方法的设计界面。

【范例 16】

使用 OR 对象设计器创建一个到 db_Books 数据库中 Books 数据表的映射关联，并显示该表中的图书信息。主要步骤如下所示。

（1）在 VS 2012 中打开应用程序的【添加新项】对话框。选择其中的【LINQ to SQL 类】项，定义名称为 dbBooks.dbml，最后单击【添加】按钮，如图 11-6 所示。

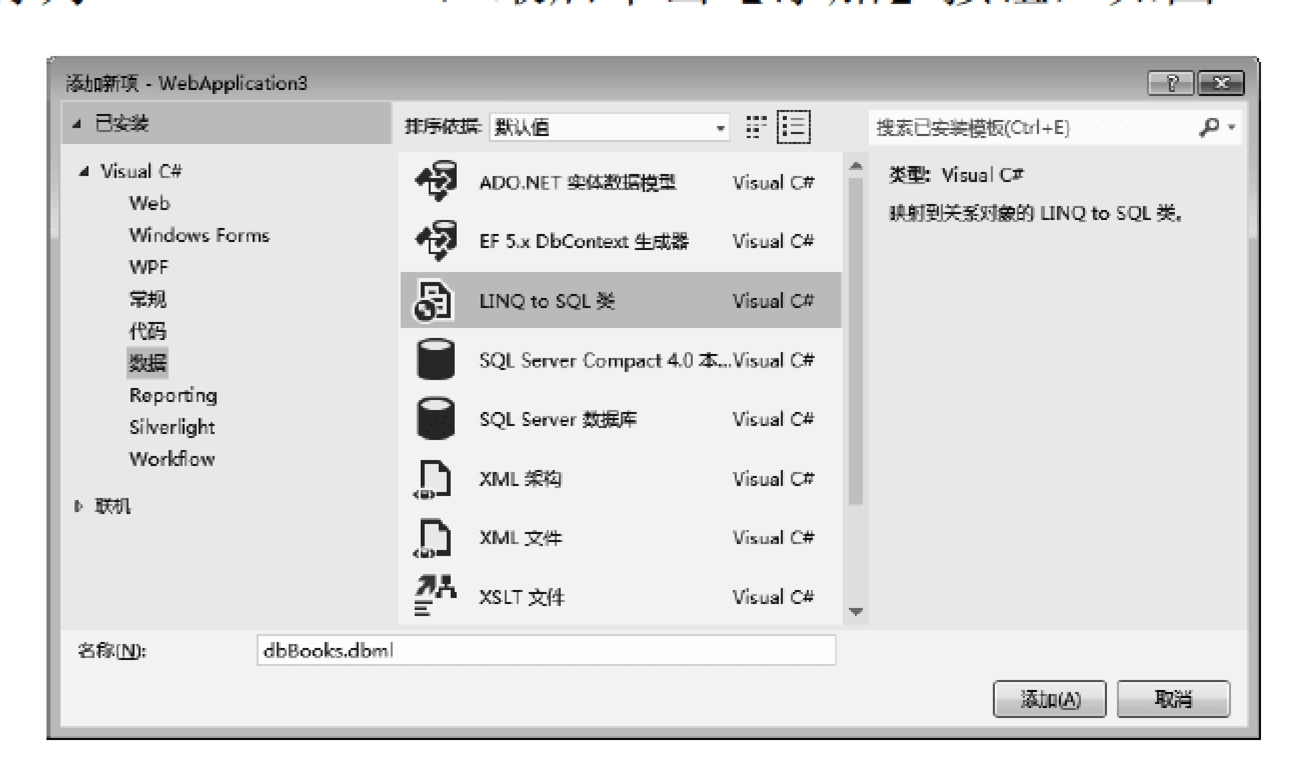


图 11-6 【添加新项】对话框

（2）打开【服务器资源管理器】窗格创建一个到 SQL Server 服务器 db_Books 数据库的连接。然后展开【表】节点，将 Books 表选中拖动到右侧的实体窗格。此时会自动生成名为 Books 的实体，将 Books 表中的列生成为实体属性，如图 11-7 所示。

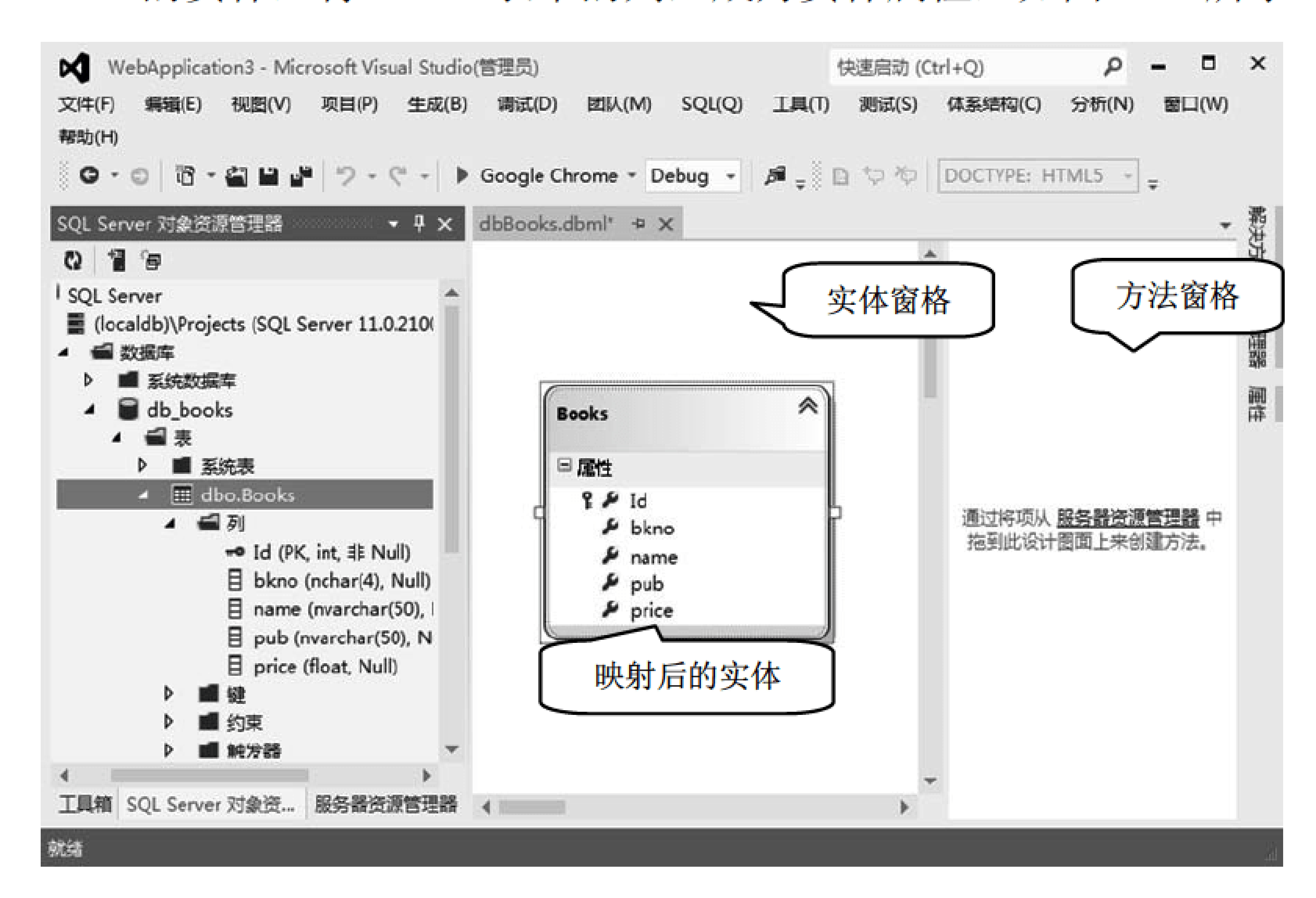


图 11-7　OR 设计器效果图

提 示

通过将数据表从【服务器资源管理器】或者【数据库资源管理器】窗格拖动到 OR 设计器上可以创建基于这些表的实体类，并由 OR 设计器自动生成代码。

（3）在 Web 窗体中添加一个 GridView 控件。在后台 Load 事件中使用 LINQ 查询数据源（Books 数据表）中的信息并绑定到 GridView 控件显示。代码如下。

```
protected void Page Load(object sender, EventArgs e)
{
    dbBooksDataContext db = new dbBooksDataContext();
                                                    //初始化 dbBooksDataContext 类
    var books = from b in db.Books
                select new
                {
                    编号 = b.Id,
                    书号 = b.bkno,
                    名称 = b.name,
                    出版社 = b.pub,
                    价格 = b.price
                };                                  //创建一个 LINQ 查询
    GridView1.DataSource = books;                   //为 GridView 控件绑定数据
    GridView1.DataBind();
}
```

（4）执行程序，在 Web 页面上会显示出 Books 数据表中的图书信息，如图 11-8 所示。



编号	书号	名称	出版社	价格
1	1001	Java现代软件开发技术	清华大学出版社	98
2	1002	XML编程技术大全	电子工业出版社	78.5
3	1003	Oracle编程入门经典	电子工业出版社	42.8
4	1004	SQL实用参考手册	清华大学出版社	39.9
5	1005	Flash动画特效	人民邮电出版社	49.5
6	1006	网页设计三剑客	清华大学出版社	24.8
7	1007	VFP实用教程	人民邮电出版社	46.6

图 11-8　显示图书信息运行效果

11.3.2　DataContext 类简介

DataContext 类是一个 LINQ to SQL 类，它在 SQL Server 数据库与映射的数据库实体类之间建立了管道。DataContext 类包含用于连接数据库以及操作数据库数据的连接字符串信息和方法。

DataContext 类构造函数有很多，常用的构造函数形式如下。

```
public DataContext(IDbConnection connection)
```

这种形式将使用连接对象创建 DataContext 类的实例。

```
public DataContext(string fileOrServerOrConnection)
```

这种形式使用连接字符串、数据库所在的服务器的名称（将使用默认数据库）或数据库所在文件的名称创建 DataContext 类的实例。

```
public DataContext(IDbConnection connection,MappingSource mapping),
```

这种形式将使用连接对象和映射源创建 DataContext 类的实例。

```
public DataContext(string fileOrServerOrConnection,MappingSource mapping)
```

使用连接字符串、数据库所在的服务器的名称（将使用默认数据库）或数据库所在文件的名称和映射源创建 DataContext 类的实例。

1. DataContext 类方法

DataContext 类包含多个可以调用的方法，例如用于将已更新数据从 LINQ to SQL 类提交到数据库的 SubmitChanges()方法。如表 11-4 所示列出了 DataContext 类的常用方法和说明。

表 11-4　DataContext 类常用方法

方法	说明
DatabaseExists()	检测指定的数据库是否存在，如果存在，则返回 true，否则返回 false
CreateDatabase()	在 DataContext 类的实例的连接字符串指定的服务器上创建数据库
DeleteDatabase()	删除 DataContext 类的实例的连接字符串标识的数据库
ExecuteCommand()	执行指定的 SQL 语句，并通过该 SQL 语句来操作数据库
ExecuteQuery()	执行指定的 SQL 查询语句，并通过 SQL 查询语句检索数据，查询结果保存数据类型为 IEnumerable 或 IEnumerable<TResult>的对象
SubmitChanges()	计算要插入、更新或删除的已修改对象的集，并执行相应的修改提交到数据库，并修改数据库
GetCommand()	获取指定查询的执行命令的信息
GetTable()	获取 DataContext 类的实例的表的集合
GetChangeSet()	获取被修改的对象，它返回由三个只读集合（Inserts、Deletes 和 Updates）组成的对象
Translate()	将 DbDataReader 对象中的数据转换为数据类型为 IEnumerable<TResult>或 IMultipleResults 或 IEnumerable 的新对象
Refresh()	刷新对象的状态，刷新的模式由 RefreshMode 枚举的值指定。该枚举包含三个枚举值：KeepCurrentValues、KeepChanges 和 OverwriteCurrentValues

例如，下面的示例调用 DatabaseExists()方法判断 Northwind 数据库是否存在，如果存在则调用 DeleteDatabase()方法删除该数据库；否则就调用 CreateDatabase()方法创建该数据库。


```
//检测 Northwind 数据库是否存在
if (db.DatabaseExists()){
    Response.Write("Northwind 数据库存在");
    db.DeleteDatabase();              //删除数据库
}
else {
    Response.Write("Northwind 数据库不存在");
    db.CreateDatabase();              //创建数据库
}
```

2. DataContext 类属性

在表 11-5 中列出了 DataContext 类的常用属性及其说明。

表 11-5　DataContext 类常用属性

属性	说明
Connection	可以获取 DataContext 类的实例的连接（类型为 DbConnection）
Transaction	为 DataContext 类的实例设置访问数据库的事务。其中，LINQ to SQL 支持三种事务：显式事务、隐式事务和显式可分发事务
CommandTimeout	可以设置或获取 DataContext 类的实例的查询数据库操作的超时期限。该时间的单位为 s，默认值为 30s
ChangeConflicts	返回 DataContext 类的实例调用 SubmitChanges()方法时导致并发冲突的对象的集合
DeferredLoadingEnabled	可以设置或获取 DataContext 类的实例是否延时加载关系
LoadOptions	可以获取或设置 DataContext 类的实例的 DataLoadOptions
Log	可以将 DataContext 类实例的 SQL 查询或命令显示在网页或控制台中

例如，下面的示例使用 Log 属性在 SQL 代码执行前在控制台窗口中显示此代码。可以将此属性与查询、插入、更新和删除命令一起使用。

```
//关闭日志功能
//db.Log = null;
//使用日志功能：日志输出到控制台窗口
db.Log = Console.Out;
var q = from c in db.Customers
     where c.City == "London"
     select c;
//日志输出到文件
StreamWriter sw = new StreamWriter(Server.MapPath("log.txt"), true);
db.Log = sw;
var q = from c in db.Customers
      where c.City == "London"
      select c;
sw.Close();
```

11.3.3　SubmitChanges()方法简介

除了使用 LINQ 对数据表进行查询外，LINQ to SQL 还可以对表中的数据进行添加、更新和删除，这与 SQL 中的 INSERT、UPDATE 和 DELETE 语句实现的功能相同。

在使用本地数据时无论对象做了多少项更改，都只是在更改内存中的副本，并未对数据库中的实际数据做任何更改。直到对 DataContext 显式调用 SubmitChanges()方法，所做的更改才会传输到服务器。DataContext 会设法将所做的更改转换为等效的 SQL 命令。

也可以使用自定义逻辑来重写这些操作，但提交顺序是由 DataContext 的一项称作“更改处理器”的服务来协调的。事件的顺序如下。

（1）当调用 SubmitChanges()方法时，LINQ to SQL 会检查已知对象的集合以确定新实例是否已附加到它们。如果已附加，这些新实例将添加到被跟踪对象的集合。

（2）所有具有挂起更改的对象将按照它们之间的依赖关系排序成一个对象序列。如果一个对象的更改依赖于其他对象，则这个对象将排在其依赖项之后。

（3）在即将传输任何实际更改时，LINQ to SQL 会启动一个事务来封装由各条命令组成的系列。

（4）对对象的更改会逐个转换为 SQL 命令，然后发送到服务器。

注 意

如果数据库检测到任何错误，都会造成提交进程停止并引发异常。将回滚对数据库的所有更改，就像未进行过提交一样。

11.3.4　插入数据

使用 LINQ to SQL 向数据库中插入数据时必须完成三个步骤：首先要创建一个包含要提交的列数据的新对象，其次将这个新对象添加到与数据库中的目标表关联的 LINQ to

SQL Table 集合，最后将更改提交到数据库。此时，LINQ to SQL 会将在应用程序中所做的更改转换成相应的 INSERT 语句。

【范例 17】

在 11.3.1 节的基础上创建一个新的图书信息对象，并将该对象添加到集合中，最后将所做的更改提交到数据库中，使之成为 Books 表中的一个新行。

实现代码如下。

```
private void Page_Load(object sender, EventArgs e)
{
    dbBooksDataContext db = new dbBooksDataContext();
                                              //初始化 dbBooksDataContext 类
    var newBook = new Books {                 //设置新行中各列的值
        Id = 8,
        bkno = "1008",
        name = "Java 测试",
        pub = "人民邮电",
        price = 39
    };
    db.Books.InsertOnSubmit(newBook);
    try {
        db.SubmitChanges();
    }
    catch (Exception ex) {
        Response.Write("出错了。信息：" + ex.ToString());
    }
}
```


上述代码首先创建了一个表示 Books 表中所有数据的 DataContext 类 db，然后创建了一个 Books 类对象 newBook 表示要新增的图书信息。然后调用 InsertOnSubmit()方法将 newBook 插入到 db 中，最后调用 SubmitChanges()方法将修改后的集合提交到 Books 表。如图 11-9 所示为添加后的 Books 表内容。



编号	书号	名称	出版社	价格
1	1001	Java现代软件开发技术	清华大学出版社	98
2	1002	XML编程技术大全	电子工业出版社	78.5
3	1003	Oracle编程入门经典	电子工业出版社	42.8
4	1004	SQL实用参考手册	清华大学出版社	39.9
5	1005	Flash动画特效	人民邮电出版社	49.5
6	1006	网页设计三剑客	清华大学出版社	24.8
7	1007	VFP实用教程	人民邮电出版社	46.6
8	1008	Java测试	人民邮电	39

图 11-9　添加后的 Books 表内容

11.3.5　更新数据

使用 LINQ to SQL 更新数据库中的数据需要三个步骤：首先要创建一个包含所有数

据的集合，然后在集合中对数据进行更改，最后将更改提交到数据库。此时，LINQ to SQL 会将所有更改转换成相应的 UPDATE 语句。

【范例 18】

在 11.3.4 节的基础上对图书进行如下更改.

（1）将编号为 5 的图书名称修改为“三维动画设计”，价格修改为 58。

（2）将所有出版社为“清华大学出版社”的图书价格修改为 100。

实现代码如下。

```
private void btnUpdate Click(object sender, EventArgs e)
{
    dbBooksDataContext db = new dbBooksDataContext();
                                        //初始化 dbBooksDataContext 类
    Books aBook = (from b in db.Books
                   where b.Id == 5
                   select b).First();    //查询编号为 5 的图书
    aBook.name = "三维动画设计";          //修改图书名称
    aBook.price = 58;                    //修改图书价格
    var upBooks = from b in db.Books
                  where b.pub == "清华大学出版社"
                  select b;
    foreach (var b in upBooks)    {
        b.price = 100;                   //批量修改图书价格
    }
    try   {
        db.SubmitChanges();
    }
    catch (Exception ex)     {
        Response.Write("出错了。信息：" + ex.ToString());
}
```

如图 11-10 所示为更新后的 Books 表内容。



编号	书号	名称	出版社	价格
1	1001	Java现代软件开发技术	清华大学出版社	100
2	1002	XML编程技术大全	电子工业出版社	78.5
3	1003	Oracle编程入门经典	电子工业出版社	42.8
4	1004	SQL实用参考手册	清华大学出版社	100
5	1005	三维动画设计	人民邮电出版社	58
6	1006	网页设计三剑客	清华大学出版社	100
7	1007	VFP实用教程	人民邮电出版社	46.6
8	1008	Java测试	人民邮电	39

图11–10 更新后的 Books 表内容

11.3.6 删除数据

如果要删除数据表中的数据，可以通过将表从对应的 LINQ to SQL 对象集合中删除来实现。此时，LINQ to SQL 会将更改转换为相应的 DELETE 语句。

【范例 19】

在 11.3.5 节的基础上对图书进行如下更改。

（1）删除编号为 5 的图书信息。

（2）删除所有价格为 100 的图书信息。

实现代码如下。

```
dbBooksDataContext db = new dbBooksDataContext();
                                          //初始化 dbBooksDataContext 类
Books delBook = (from b in db.Books
              where b.Id == 5
              select b).First();
db.Books.DeleteOnSubmit(delBook);         //删除编号为 5 的图书信息
var delBooks = from b in db.Books
             where b.price == 100
             select b;
foreach(var b in delBooks){
   db.Books.DeleteOnSubmit(b);            //批量删除
}
try{
   db.SubmitChanges();
}
catch (Exception ex){
   Response.Write("出错了。信息：" + ex.ToString());
}
```


如图 11-11 所示为删除后的 Books 表内容。



编号	书号	名称	出版社	价格
2	1002	XML编程技术大全	电子工业出版社	78.5
3	1003	Oracle编程入门经典	电子工业出版社	42.8
7	1007	VFP实用教程	人民邮电出版社	46.6
8	1008	Java测试	人民邮电	39

图11-11　删除后的 Books 表内容

注 意

LINQ to SQL 不支持且无法识别级联删除操作。如果要在对行有约束的表中删除行，则必须在数据库的外键约束中设置 ON DELETE CASCADE 规则，或者使用自己的代码首先删除防止删除父对象的子对象，否则会引发异常。

11.4 实验指导——多表关联查询

通过使用点表示法在查询时利用对象关系来访问关系属性，以及从一个对象定位到另一个对象。这些关系引用相当于数据库中的外键关系。使用这些关系的操作会转换成等效的 SQL 多表联接查询。

在 LINQ to SQL 中创建表间关系属性非常简单，只需要在表中设置好表之间的主外键关系，那么在 OR 设计器中就会自动生成这种关系属性。下面通过一个案例介绍 LINQ to SQL 的多表查询。

在 11.3.1 节 Books 表所在数据库 db_Books 中增加一个表示图书分类的 Types 表。该表包含一个 int 类型的 tid 列，一个 nvarchar 类型的 tname 列。对学生信息表 Books 进行修改，添加一个名为 tid 的列。然后进行如下操作。

（1）打开 11.3.1 节的 OR 设计器首先将 Books 实体删除，再将 Books 表和 Types 表添加到实体设计器。

（2）在 OR 设计器空白处右击，选择【添加】|【关联】命令，在弹出的【关联编辑器】对话框中对关联进行编辑，如图 11-12 所示。如图 11-13 所示为关联后的实体图。

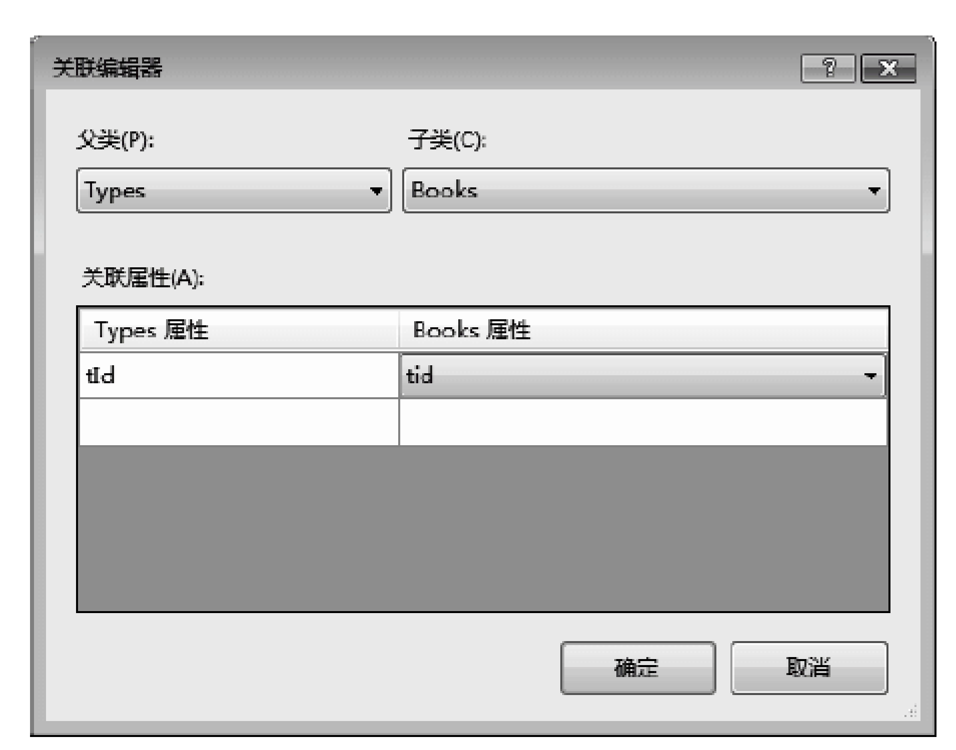


图11-12 【关系编辑器】对话框

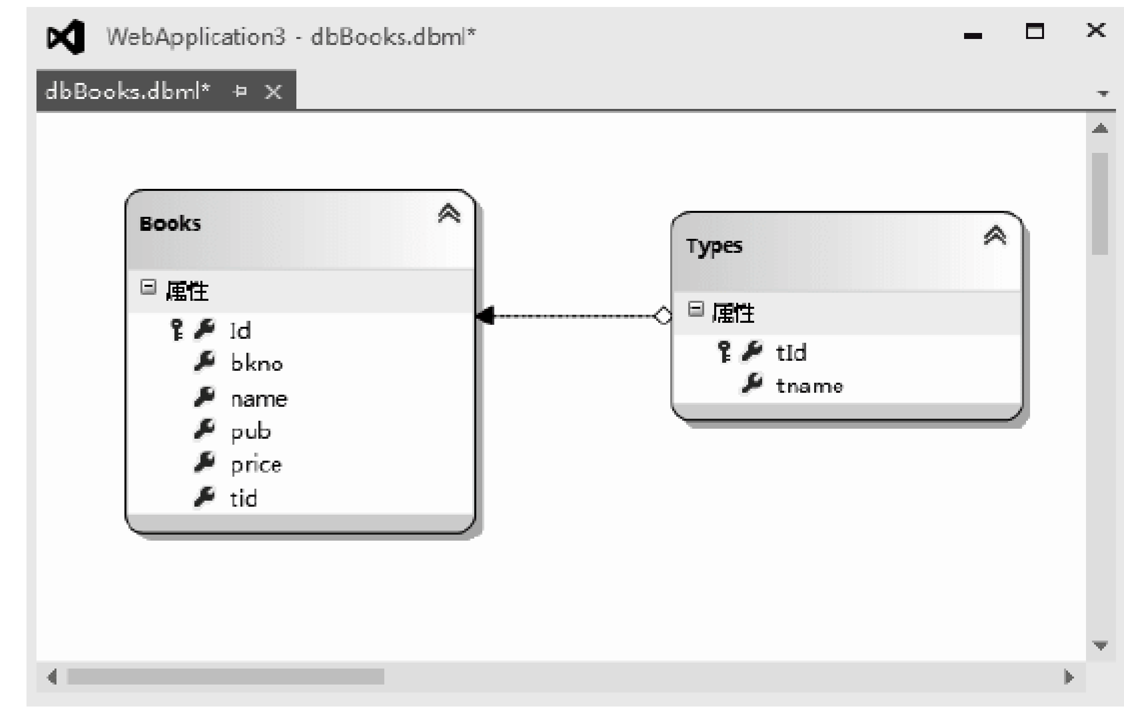


图11-13 带关联的实体图

（3）创建一个 Web 窗体并添加 GridView 控件。然后使用 LINQ to SQL 查询出图书编号、书号、名称、出版社和价格，以及对应的分类名称。实现代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    dbBooksDataContext db = new dbBooksDataContext();
                                        //初始化 dbBooksDataContext 类
    var books = from b in db.Books
                select new
                {
                    编号 = b.Id,
                    书号 = b.bkno,
                    名称 = b.name,
                    出版社 = b.pub,
                    价格 = b.price,
                    分类名称=b.Types.tname      //关联 Types 表的 tname 列
                };                          //创建一个 LINQ 查询
    GridView1.DataSource = books;
    GridView1.DataBind();
}
```

（4）在浏览器中打开 Web 页面，运行效果如图 11-14 所示。



编号	书号	名称	出版社	价格	分类名称
2	1002	XML编程技术大全	电子工业出版社	78.5	其他
3	1003	Oracle编程入门经典	电子工业出版社	42.8	数据库开发
7	1007	VFP实用教程	人民邮电出版社	46.6	数据库开发
8	1008	Java测试	人民邮电	39	程序设计

图11-14 查看带分类的图书信息效果

思考与练习

一、填空题

1. 在.NET Framework 中与 LINQ 相关的类都在__________命名空间下。

2. __________类通过对 IQueryable<T>提供扩展方法，实现 LINQ 标准查询运算符。

3. 在使用 join 进行内联接时 equals 关键字左侧必须是__________子句的数据源。

4. 使用 DataContext 类的__________方法判断一个数据库是否存在。

5. DataContext 类的__________属性可以获取连接字符串。

二、选择题

1. 下列选项中，不能使用 LINQ 进行查询的是__________。

A. ICompare<T>接口

B. IEnumerable<T>接口

C. IQueryable<T>接口

D. Enumberable 类

2. 下列不属于 LINQ 查询表达式子句的是__________。

A. select 子句

B. from 子句

C. create 子句

D. join 子句

3. 假设有如下的数据源，下面 LINQ 查询表达式不正确的是__________。

```
int[] numbers ={ 1, 8, 2, 5,6,3,
3,7,10 };
```

A. from num in numbers select num;

B. from num in numbers where num>4 && num+1>5 select num;

C. from num in numbers orderby num descending select num;

D. from num in numbers group by num select num;

4. 使用 join 子句不能实现的联接方式是__________。

A. 左外联接

B. 内联接

C. 外联接

D. 分组联接

5. 调用 DataContext 类的__________方法可以实现对数据表的删除操作。

A. SubmitChanges()

B. Submit()

C. Delete()

D. DeleteData()

三、简答题

1. 简述 LINQ 查询表达式的组成部分。

2. 简述使用查询表达式对数据源进行条件筛选的方法。

3. 举例说明 orderby 子句在 LINQ 中的应用。

4. 简述 OR 设计器对实体的影响。

5. 简述 DataContext 类的作用。

第 12 章　高级技术应用

ASP.NET 的功能非常强大，除了前面介绍的内容外，本章将介绍一些高级的技术。例如，如何在 ASP.NET 中实现文件的上传和下载操作，如何对磁盘下的目录和文件进行添加、删除和移动等操作，以及两种第三方控件的使用等内容。

本章学习要点：

- ❑ 掌握 FileUpload 控件的使用
- ❑ 掌握如何实现文件下载
- ❑ 熟悉 FileInfo 类的常用属性
- ❑ 掌握 File 类和 FileInfo 类的使用
- ❑ 熟悉 DirectoryInfo 类的常用属性
- ❑ 掌握 Directory 和 DirectoryInfo 类的使用
- ❑ 掌握 AspNetPager 控件的使用
- ❑ 熟悉 SerialNumber 控件的使用

12.1 文件上传与下载

文件上传与下载是开发者经常实现的操作，如用户修改个人资料时需要更改个人头像，将更改后的文件重新上传到服务器。本节将分别介绍文件上传与文件下载。

12.1.1 文件上传

ASP.NET 中提供了一个 FileUpload 控件，该控件显示一个文本框和一个【浏览】按钮，使用户可以选择客户端上的文件并将它上传到 Web 服务器。用户通过在控件的文本框中输入本地计算机上文件的完整路径（如 C:\MyFiles\TestFile.txt）来指定要上载的文件。也可以通过单击【浏览】按钮，然后在【选择文件】对话框中定位文件来选择文件。

用户选择要上传的文件后，FileUpload 控件不会自动将该文件保存到服务器。开发者必须显式提供一个控件或机制，使用户能提交指定的文件。如可以提供一个按钮，用户单击它即可上载文件。为保存指定文件所写的代码应调用 SaveAs()方法，该方法将文件内容保存到服务器上的指定路径。通常，在引发回发到服务器的事件处理方法中调用 SaveAs()方法。

除了 SaveAs()方法外，FileUpload 控件提供了一系列的属性，常用属性如表 12-1 所示。

表 12-1 FileUpload 控件的常用属性

属性名称	说明
FileBytes	从使用 FileUpload 控件指定的文件中获取一个字节数组
FileContent	获取 Stream 对象，它指向要使用 FileUpload 控件上载的文件
FileName	获取客户端上使用 FileUpload 控件上载的文件的名称
HasFile	获取一个值，该值指示 FileUpload 控件是否包含文件
PostedFile	获取使用 FileUpload 控件上载的文件的基础 HttpPostedFile 对象

在表 12-1 中，PostedFile 属性返回 HttpPostedFile 对象，该对象提供对客户端已上载的单独文件的访问，它的常用属性如下。

（1）ContentLength：获取上载文件的大小（以字节为单位）。

（2）ContentType：获取客户端发送的文件的 MIME 内容类型。

（3）FileName：获取客户端上的文件的完全限定名称。

（4）InputStream：获取一个 Stream 对象，该对象指向一个上载文件，以准备读取该文件的内容。

【范例 1】

完成用户注册功能，通过 FileUpload 控件选择用户头像，然后将选择的头像上传到指定的服务器，并保存到数据库中。步骤如下。

（1）创建一个模拟实现用户注册的页面，在表单中添加七行两列的表格，前三行提供用户名输入框、联系电话输入框和用户头像选择，第四行提供操作按钮。主要代码如下。

```
<asp:TextBox  ID="txtName"  runat="server"></asp:TextBox><asp:Required
FieldValidator ID="rfvName" runat="server" ControlToValidate="txtName">
必须输入</asp:RequiredFieldValidator>
<asp:TextBox ID="txtPhone" runat="server" TextMode="Phone"></asp:Text
Box><asp:RequiredFieldValidator ID="rfvPhone" runat="server" ControlTo
Validate="txtPhone">必须输入</asp:RequiredFieldValidator>
<asp:FileUpload ID="fuUploadFile" runat="server" />
<asp:Button  ID="btnUpload"  runat="server"  Text=" 上 传 "  OnClick=
"btnUpload_Click" />
```

（2）在表格的后三行中，都包含一个 Label 控件，第一个 Label 控件用于显示结果，第二个 Label 显示上传文件的信息，第三个 Label 显示图像。代码如下。

```
<asp:Label ID="lblResult" runat="server"></asp:Label>
<asp:Label ID="lblInfo" runat="server"></asp:Label>
<asp:Image ID="imgShow" runat="server" Width="300" Height="200" />
```

（3）为页面中的 Button 控件添加 Click 事件代码，首先获取用户在页面中输入的用户名和联系电话。代码如下。

```
protected void btnUpload_Click(object sender, EventArgs e) {
    string userName = txtName.Text;         //获取用户名
    string userPhone = txtPhone.Text;       //获取电话
    /* 省略其他代码 */
```

```
}
```

（4）通过 FileUpload 控件的 HasFile 属性判断是否已有选择文件，如果是则通过 ContentType 属性获取文件内容类型，如果类型符合要求，则获取上传文件的路径、文件名称和服务器端指定文件的路径。代码如下。

```
if (fuUploadFile.HasFile)   {                        //判断是否选择文件
    string fileContentType = fuUploadFile.PostedFile.ContentType;
                                                     //获取文件内容类型
    if (fileContentType == "image/bmp" || fileContentType == "image/gif"
    || fileContentType == "image/pjpeg"){            //判断类型是否符合条件
        string name = fuUploadFile.PostedFile.FileName; //客户端文件路径
        FileInfo file = new FileInfo(name);
        string fileName = file.Name;                 //文件名称
        string webFilePath = Server.MapPath("fileupload/" + fileName);
                                                     //服务器端文件路径
        /* 省略其他代码 */
    } else {
        lblResult.Text = "提示：文件类型不符";
    }
}
```


（5）通过 File.Exists()方法判断 webFilePath 路径中是否已经存在文件，如果不存在则调用 SaveAs()方法保存文件，并显示文件信息。然后再调用 SqlHelper 类中的 ExecuteNonQuery()方法将用户输入的数据提交到数据库。代码如下。

```
if (!File.Exists(webFilePath)) {                //判断相同文件是否存在
    try {
        fuUploadFile.SaveAs(webFilePath);       //使用 SaveAs()方法保存文件
        lblResult.Text = "提示：文件" + fileName + "上传成功！路径是：" +
        "fileupload/" + fileName;
        lblInfo.Text = "文件大小："+file.Length +"字节<br/>文件扩展名：" +
         file.Extension + "<br/>";
        imgShow.ImageUrl = "fileupload/" + fileName;
        string sql = "INSERT INTO RegisterUserInfo (UserName,UserPhone,
         UserImage) VALUES (@name,@phone,@image)";
        SqlParameter[] para = new SqlParameter[]{
            new SqlParameter("@name",userName),
            new SqlParameter("@phone",userPhone),
            new SqlParameter("@image","~/UploadDown/fileupload/"+fileName)
        };
        SqlHelper.ExecuteNonQuery(SqlHelper.connString, CommandType. Text,
        sql, para);
    } catch (Exception ex) {
        lblResult.Text = "提示：文件上传失败，失败原因：" + ex.Message;
    }
} else {
    lblResult.Text = "提示：文件已经存在，请重命名后上传";
```

```
    imgShow.ImageUrl = "fileupload/" + fileName;
}
```

（6）运行页面输入内容后单击按钮进行测试，如图 12-1 所示。



图 12-1　文件上传

> **注 意**
>
> 实现文件上传时最好使用 IE 10 及其版本以下的浏览器，因为在这个浏览器中 FileUpload 控件的 FileName 属性能够获取到完整的路径。如果 IE 浏览器禁用，不包含上传路径，那么可以在【Internet 选项】|【安全】|Internet|【自定义设置】里启用将文件上传到服务器时包含本地目录路径。

在文件上传的过程中，文件数据作为页面请求的一部分，上传并缓存到服务器的内存中，然后再写入服务器的物理硬盘中。实现文件上传时，需要注意以下三点。

1．确认是否包含文件

在调用 SaveAs()方法将文件保存到服务器之前，通过 HasFile 属性验证 FileUpload 控件是否确实包含文件。如果 HasFile 属性的返回值为 true，则调用 SaveAs()方法。

2．文件上传大小限制

默认情况下，上传文件大小限制为 4096KB，即 4MB。可以通过设置 httpRuntime 元素的 maxRequestLength 属性来允许上传更大的文件。如果要增加整个应用程序所允许的最大文件大小，需要设置 Web.config 文件中的 maxRequestLength 属性。如果要增加指定页所允许的最大文件大小，需要设置 Web.config 文件中 location 元素内的 maxRequestLength 属性。

3．上传文件夹的写入访问权限

应用程序可以通过两种方式获取写入访问权限。开发者可以将要保存上传文件的目录的写入访问权限显式授予运行应用程序所使用的账户；也可以提高为 ASP.NET 应用程序授予的信任级别。

12.1.2 文件下载

文件下载是与文件上传相反的过程。文件上传是将文件从客户端保存到服务器端，而文件下载则将文件从服务器端下载到客户端。文件下载主要通过 Response 对象的相关属性和方法来实现。主要步骤如下。

（1）通过 ContentType 属性设置输出流的类型。

（2）调用 AddHeader()方法定义文件下载中的应答头。

（3）执行文件下载操作。

（4）释放资源。

【范例 2】

获取当前程序目录下 fileupload 文件夹中的所有文件信息，包括文件名称、扩展名、文件大小和上传时间，然后单击文件的链接进行下载。步骤如下。

（1）在页面中添加 Repeater 控件，为该控件添加 ItemTemplate 模板。代码如下。

```
<asp:Repeater ID="Repeater1" runat="server">
   <ItemTemplate>
      <table width="90%" align="center" border="1" cellpadding="1"
      cellspacing="0" bgcolor="#e1e1e1" class="title font">
         <tr>
            <td class="title_font" width="10%" align="center">文件名称：
            </td>
            <td><asp:Label ID="FileTitle" runat="server" Text='<%#Eval
             ("Name") %>'></asp:Label></td>
            <td width="10%" align="center">扩展名：</td>
            <td width="6%" align="center"><%#Eval("Extension") %></td>
            <td width="10%" align="center">文件大小：</td>
            <td width="8%" align="center"><%#Eval("Length") %>KB</td>
            <td width="10%" align="center">上传时间：</td>
            <td><%#Eval("CreateTime") %></td>
            <td width="10%" colspan="2" align="center"><asp:LinkButton
            ID="LinkButton1" CommandArgument='<%#Eval("Name") %>'
            runat= ="server" OnClick="LinkButton1_Click">下载文件
            </asp:LinkButton></td>
         </tr>
      </table>
   </ItemTemplate>
</asp:Repeater>
```

（2）在页面的 Load 事件中添加代码，页面首次加载时获取 fileupload 文件夹下的所有文件，将绑定到 Repeater 控件。代码如下。

```
protected void Page Load(object sender, EventArgs e) {
    if (!Page.IsPostBack) {                        //如果首次加载
        string downpath = Server.MapPath("fileupload");
                                                   //返回指定的文件路径
        DirectoryInfo dirinfo = new DirectoryInfo(downpath);
                                                   //创建 DirectoryInfo 对象
        if (!dirinfo.Exists)                                   //如果目录不存在
            ClientScript.RegisterStartupScript(GetType(),"","<script>
           alert('文件目录不存在')</script>");
        else {
            FileInfo[] filist = dirinfo.GetFiles(); //获取该目录下的所有文件
            IList<Down> downinfo = new List<Down>();     //文件列表集合对象
            foreach (FileInfo fi in filist) {            //遍历列表对象
            Down di = new Down(fi.Name, fi.Extension, fi.Length.ToString(),
            fi.CreationTime);
                downinfo.Add(di);
            }
            Repeater1.DataSource = downinfo;             //指定数据源
            Repeater1.DataBind();
        }
    }
}
```

在上述代码中，DirectoryInfo 类专门针对目录处理，FileInfo 类专门针对文件处理。Down 是文件下载类，包含对文件名称、文件扩展名、文件大小和上传时间等字段的封装。

（3）为 Repeater 控件中的 LinkButton 控件添加 Click 事件，在事件代码中获取用户要下载的文件，然后通过 File.Exists()方法判断文件是否存在，如果存在则下载。代码如下。

```
protected void LinkButton1 Click(object sender, EventArgs e) {
    string downFile = ((LinkButton)sender).CommandArgument;
    string path = Server.MapPath("fileupload") + "\\" + downFile;
                                                   //服务器端下载文件的路径
    if (File.Exists(path)) {
        FileInfo fi = new FileInfo(path);
        Response.ContentEncoding = System.Text.Encoding. GetEncoding
        ("UTF-8");                                 //解决中文乱码
        Response.AddHeader("Content-Disposition","attachment;filename=
        "+Server.UrlEncode(fi.Name));              //将 HTTP 头添加到输出流
        Response.AddHeader("Content-length", fi.Length.ToString());
        Response.ContentType = "application/octet-stream";
                                                   //设置输出流的类型
        Response.WriteFile(fi.FullName);
```

```
                        //将指定文件的内容作为文件块直接写入 HTTP 响应输出流
            Response.End();
        } else
            Page.ClientScript.RegisterStartupScript(GetType(), "", "<script>
            alert('你要下载的文件不存在,可以地址发生改变。请确认后下载!')</script>");
    }
```

在上述代码中，通过 Response 对象的 AddHeader()方法设置 HTTP 标头名称和值。ContentType 属性用于设置输出流的类型。WriteFile()方法表示将指定文件的内容写入到 HTTP 输出流中。

（4）运行页面查看效果，如图 12-2 所示。



图 12-2 文件下载页面

（5）单击图 12-2 中某个文件后名称为“下载文件”链接进行下载，效果图不再显示。

【范例 3】

范例 2 通过 WriteFile()方法进行下载，除了这种方法外，还可以通过其他方式进行下载。如下代码以字符流的形式下载文件。

```
protected void Button4 Click(object sender, EventArgs e) {
    string fileName = "pic.zip";            //客户端保存的文件名
    string filePath = Server.MapPath("DownLoad/aaa.zip");//路径
    FileStream fs = new FileStream(filePath, FileMode.Open);
    byte[] bytes = new byte[(int)fs.Length];
    fs.Read(bytes, 0, bytes.Length);
    fs.Close();
    Response.ContentType = "application/octet-stream";
    Response.AddHeader("Content-Disposition","attachment;filename=
     "+HttpUtility.UrlEncode(fileName,System.Text.Encoding.UTF8));
    Response.BinaryWrite(bytes);
    Response.Flush();
    Response.End();
}
```

无论是通过 WriteFile()方法下载文件或是字符流下载文件，都用到 AddHeader()方法设置 HTTP 标头名称和值，常设的标头的信息如表 12-2 所示。

表 12-2　AddHeader()方法常设的标头信息

标头信息	说明
Cache-Control	告诉浏览器或者其他客户，什么环境可以安全地缓存文档
Content-Disposition	要求浏览器询问客户，将响应存储在磁盘上给定名称的文件中
Content-Encoding	文档的编码（Encode）方法，只有在解码之后才可以得到 Content-Type 头指定的内容类型
Content-Language	文档使用的语言
Content-Length	表示内容长度，当浏览器使用持久 HTTP 连接时需要这个数据
Content-Type	表示后面的文档属于什么 MIME 类型

12.2　文件操作

细心的用户可以发现，12.1 节介绍文件上传和下载时使用过 File 和 FileInfo 类，这两个类专门对文件进行操作，如判断文件是否存在、创建文件、复制文件和移动文件等。但是不同的是：File 是一个静态类，FileInfo 是一个密封的实例类，不能被继承。

12.2.1　获取文件基本信息

FileInfo 类提供了许多属性获取文件的基本信息，如表 12-3 所示。

表 12-3　FileInfo 类的常用属性

属性名称	说明
Attributes	获取或设置当前 FileSystemInfo 的 FileAttributes 属性
CreationTime	获取或设置当前 FileSystemInfo 对象的创建时间
CreationTimeUtc	获取或设置当前 FileSystemInfo 对象的创建时间，其格式为通用 UTC 时间
Directory	获取父目录的实例
DirectoryName	获取表示目录的完整路径的字符串
Exists	获取指示文件是否存在的值
Extension	获取表示文件扩展名部分的字符串
FullName	获取目录或文件的完整目录
IsReadOnly	获取或设置确定当前文件是否为只读的值
LastAccessTime	获取或设置上次访问当前文件或目录的时间
LastAccessTimeUtc	获取或设置上次访问当前文件或目录的时间，其格式为通用 UTC 时间
LastWriteTime	获取或设置上次写入当前文件或目录的时间
LastWriteTimeUtc	获取或设置上次写入当前文件或目录的时间，其格式为通用 UTC 时间
Length	获取当前文件的大小
Name	获取文件名

【范例 4】

获取当前程序下 Images 文件夹下 bullet.png 文件的详细信息，如文件名称、文件大小、最后一次更新时间、最后一次访问时间和文件的完整目录等。后台代码如下。

```
protected void Page_Load(object sender, EventArgs e) {
    string filepath = Server.MapPath("~") + "Images\\bullet.png";
    FileInfo fi = new FileInfo(filepath);   //实例化 FileInfo 对象
    if (fi.Exists)                          //如果用户输入的文件路径存在
```

```
            lblResult.Text = "文件名称: " + fi.Name + "<br/>文件大小: " + fi.Length
            + " 字节<br/>最后一次更新时间:"+fi.LastWrite Time.ToLongDateString()
            +fi.LastWriteTime.ToLongTimeString() + "<br/>最后一次访问时间: " +
            fi.LastAccessTime.ToLongDateString()+fi.LastAccessTime.ToLongTime
            String() + "<br/>是否为只读文件: " + fi.IsReadOnly.ToString() + "<br/>
            文件的完整目录: " + fi.FullName + "<br/>文件扩展名: " + fi.Extension +
            "<br/>目录的完整路径: " + fi.DirectoryName + "<br/>文件的创建时间: " +
            fi.CreationTime.ToLongDateString() + fi.CreationTime.ToLong Time
            String();
        else
            lblResult.Text = "很抱歉，您输入的文件路径并不存在，请输入正确的路径。";
    }
```

12.2.2 判断文件是否存在

判断是否存在有两种方式：一种是通过 FileInfo 类的 Exists 属性，如范例 4；另一种是通过 File 类的 Exists()静态方法。

【范例 5】

直接通过 File.Exists()方法判断 D 磁盘下是否存在 mywork.txt 文件，并弹出结果。代码如下。

```
if (File.Exists(@"D:\mywork.txt"))
    Response.Write("<script>alert('该文件存在');</script>");
else
    Response.Write("<script>alert('该文件不存在')</script>");
```

12.2.3 创建文件

File 类和 FileInfo 类都提供了创建文件的方法，File.Create()方法创建文件时有以下 4 种重载方式。

```
FileStream File.Create(string path)
FileStream File.Create(string path, int bufferSize)
FileStream File.Create(string path, int bufferSize, FileOptions options)
FileSteam File.Create(string path, int bufferSize, FileOptions options,
System.Security.AccessControl.FileSecurity fileSecurity)
```

其中，path 表示文件的名称；bufferSize 用于读取和写入文件的已放入缓冲区的字节数；options 描述如何创建或覆盖该文件，它的值是 System.IO.FileOptions 的值之一；fileSecurity 确定文件的访问控制和审核安全性，它的值是 System.Security.AccessControl.FileSecurity 的值之一。

【范例 6】

如果 D 磁盘下 mywork.txt 文件不存在，那么调用 File.Create()方法创建。代码如下。

```
if (!File.Exists(@"D:\mywork.txt"))
    File.Create(@"D:\mywork.txt");
else
    Response.Write("<script>alert('该文件不存在')</script>");
```

FileInfo 类的 Create()方法可以创建或覆盖指定的文件，实例化 FileInfo 类时传入一个路径，然后直接调用 Create()方法即可。

【范例 7】

如下代码演示 FileInfo 类 Create()方法的使用。

```
FileInfo fi = new FileInfo(@"D:\mywork.txt");
fi.Create();
```

12.2.4　删除文件

File 和 FileInfo 类都提供了 Delete()方法删除文件。File 类使用 Delete()方法需要传入一个参数，该参数表示要删除的文件的名称，它不支持通配符。语法形式如下。

```
void File.Delete(string pah)                //删除指定的文件
```


FileInfo 类使用 Delete()时直接调用即可。语法形式如下。

```
void FileInfo.Delete()                  //永久删除文件
```

【范例 8】

通过 File.Exists()方法判断 D 磁盘下 mywork.txt 文件是否存在，如果存在则删除，否则弹出提示。代码如下。

```
if (File.Exists(@"D:\mywork.txt"))
    File.Delete(@"D:\mywork.txt");
else
    Response.Write("<script>alert('该文件不存在')</script>");
```

12.2.5　移动文件

移动文件是指将当前文件移动到新的位置。File 类提供 Move()方法移动文件，开发者在使用时需要传入两个参数：第一个参数表示要移动的文件的名称，第二个参数表示文件的新路径。语法如下。

```
void File.Move(string sourceFileName, string destFileName)
```

FileInfo 类提供 MoveTo()方法移动文件，开发者在使用时需要传入一个参数，该参数是指要将文件移动到的路径，可以指定另一个文件名。语法如下。

```
void FileInfo.MoveTo(string destFileName)
```

> **提 示**
> 移动文件操作实际上是删除源文件并且创建一个新的目标文件，它和复制文件操作都支持相对路径。移动文件时可以在相同或不同的磁盘下移动，但是移动目录时必须在同一个磁盘下进行。

【范例 9】

判断 D 磁盘下是否存在 mywork.txt 文件，如果存在则移动文件到 F 磁盘，否则创建文件。代码如下。

```
FileInfo fi = new FileInfo(@"D:\mywork.txt");
if (fi.Exists) {  //如果文件存在
    try {
        fi.MoveTo(@"F:\mytest\work1.txt");
    }
    catch (Exception ex) {
        Response.Write("出现错误，原因是：" + ex.Message);
    }
}else
    fi.Create();
```

12.2.6　复制文件

复制文件是指将指定文件中的内容复制到另一个文件中。File 类的 Copy()方法将现有文件复制到新文件，但是不允许覆盖同名的文件。开发者在使用时需要传入两个参数：第一个参数是要复制的文件；第二个参数是目录文件的路径，它不能是一个目录或现有文件。语法形式如下。

```
void File.Copy(string sourceFileName, string destFileName)
```

FileInfo 类的 CopyTo()方法将现有文件复制到新文件，不允许覆盖现有文件。开发者在使用 CopyTo()方法时需要传入一个参数，该参数表示要复制到新文件的名称。形式如下。

```
FileInfo FileInfo.CopyTo(string destFileName)
```

【范例 10】

如果 F 磁盘下 mytest 文件夹中的 work1.txt 文件存在，则调用 CopyTo()方法移动文件，否则创建文件。代码如下。

```
FileInfo fi = new FileInfo(@"F:\mytest\work1.txt");
if (fi.Exists) {                //如果文件存在
    try {
        fi.CopyTo(@"D:\work1.txt");
    } catch (Exception ex) {
        Response.Write("出现错误，原因是：" + ex.Message);
```

```
        }
    } else
        fi.Create();
```

12.3 目录操作

磁盘中的文件过多时会将它们分类，将相同功能的文件放在同一个目录中。在 ASP.NET 中，通常使用 Directory 类和 DirectoryInfo 类操作目录。

Directory 是一个静态类，该类提供了一系列操作目录的静态方法。DirectoryInfo 是一个实例类，调用其实例方法可以有效地对一个目录进行操作。

12.3.1 获取目录基本信息

DirectoryInfo 类包含 13 个属性，如表 12-4 所示。

表 12-4 DirectoryInfo 类的 13 个属性

属性名称	说明
Exists	判断指定路径的目录是否存在，如果存在则返回 true，否则返回 false
Name	获取目录名称
Parent	获取指定子目录的父目录名称
Root	获取目录的根部分
Attributes	获取或设置当前目录的 FileAttributes
CreationTime	获取或设置当前目录的创建时间
CreationTimeUtc	获取或设置当前目录的创建时间，其格式为 UTC 时间
Extension	获取表示文件扩展名部分的字符串
FullName	获取目录或文件的完整目录
LastAccessTime	获取或设置上次访问当前文件或目录的时间
LastAccessTimeUtc	获取或设置上次访问当前文件或目录的时间，其格式为 UTC 时间
LastWriteTime	获取或设置上次写入当前文件或目录的时间
LastWriteTimeUtc	获取或设置上次写入当前文件或目录的时间，其格式为 UTC 时间

【范例 11】

在页面后台的 Load 事件中添加代码，如果 F 磁盘下的 mytest 目录存在，则获取该目录的基本信息，如创建时间、完整路径和目录名称等。代码如下。

```
protected void Page_Load(object sender, EventArgs e) {
    DirectoryInfo di = new DirectoryInfo(@"F:\mytest");
    if (di.Exists)
        Response.Write("目录创建时间：" + di.CreationTime. ToShort Date
         String()+"<br/>完整路径：" + di.FullName + "<br/>最后一次访问时间：" 
         + di.LastAccessTime + "<br/>最后一次修改时间：" + di.LastWriteTime
         + "<br/>目录名称：" + di.Name + "<br/>父目录：" + di.Parent + "<br/>
         所在驱动器：" + di.Root.ToString());
}
```

12.3.2 判断目录是否存在

判断目录是否存在时有两种方式：一种是通过 DirectoryInfo 类的 Exists 属性，如范例 11；另一种是通过 Directory 类的 Exists()方法。Exists()方法的语法形式如下。

```
bool Directory.Exists(string path)                    //path 表示要测试的路径
```

【范例 12】

更改范例 11 的代码，通过 Directory 类的 Exists()方法判断目录是否存在。代码如下。

```
DirectoryInfo di = new DirectoryInfo(@"F:\mytest");
if (Directory.Exists(@"F;\mytest"))
    Response.Write("目录创建时间：" + di.CreationTime.ToShortDateString() +
    "<br/>完整路径：" + di.FullName + "<br/>最后一次访问时间：" + di. LastAccess
     Time + "<br/>最后一次修改时间：" + di.LastWriteTime + "<br/>目录名称：" +
     di.Name + "<br/>父目录：" + di.Parent + "<br/>所在驱动器：" +
     di.Root.ToString());
```

12.3.3 创建目录

Directory 类中的 CreateDirectory()方法可以创建目录。两种形式如下。

```
CreateDirectory(string path)
CreateDirectory(string path, DirectorySecurity directorySecurity)
```

其中，path 参数指定要创建的目录路径，它可以是绝对路径或相对路径；directorySecurity 参数应用指定的 Windows 安全性。

DirectoryInfo 类的 Create()方法直接创建目录。除了这个方法外，还可以通过 CreateSubdirectory()方法在指定的路径中创建一个或者多个子目录。CreateSubdirectory()方法的基本语法如下。

```
DirectoryInfo DirectoryInfo.CreateSubdirectory(string path)
```

其中，path 参数表示指定的路径，它不能是另一个磁盘卷或者“统一命名约定”(UNC)名称。

【范例 13】

通过 if 语句判断 E 磁盘下是否存在 MyInfoList 目录，如果存在则创建一个名称为“我的照片”的子目录，否则创建 MyInfoList 目录。代码如下。

```
DirectoryInfo di = new DirectoryInfo(@"E:\MyInfoList");
if (di.Exists)
    di.CreateSubdirectory("我的照片");
else
    di.Create();
```

12.3.4　删除目录

Directory 类的 Delete()方法可以直接删除指定的目录。它的两种形式如下。

```
void Directory.Delete(string path)      //从指定路径删除空目录
//删除指定的目录并（如果指示）删除该目录中的所有子目录和文件
void Directory.Delete(string path, bool recursive)
```

其中，path 表示要移除的空目录的名称，这个目录必须可写或者为空。recursive 是一个布尔值，如果要删除 path 中的目录、子目录和文件，则为 true；否则为 false。

DirectoryInfo 类的 Delete()方法也可以删除目录。它的两种形式如下。

```
void DirectoryInfo.Delete()
void DirectoryInfo.Delete(bool recursive)
```

【范例 14】

判断 E 磁盘下是否存在 MyInfoList 目录，如果存在直接调用 Delete()方法删除，并且删除该目录下的所有子目录和文件。代码如下。


```
DirectoryInfo di = new DirectoryInfo(@"E:\MyInfoList");
if (di.Exists)
    di.Delete(true);
```

12.3.5　遍历目录

遍历目录是指获取指定目录下的子目录和文件，如用户删除目录时可以首先遍历目录中的内容，判断该目录下是否包含其他子目录和文件。遍历目录涉及 Directory 和 DirectoryInfo 类中的多个方法，与 Directory 类有关的遍历方法如下。

（1）GetDirectories()方法：返回指定目录中子目录的名称。

（2）GetFiles()方法：返回指定目录中文件的名称。

（3）GetFileSystemEntries()方法：返回指定目录中所有文件和子目录的名称。

与 DirectoryInfo 类有关的遍历方法如下。

（1）GetDirectories()方法：返回当前目录的子目录。

（2）GetFiles()方法：返回当前目录的文件列表。

（3）GetFileSystemInfos()方法：返回表示某个目录中所有文件和子目录的强类型 FileSystemInfo 项的数组。

【范例 15】

遍历当前应用程序下 DirectoryOper 目录下所有子目录和文件。步骤如下。

（1）在表单元素中创建表格元素，代码如下。

```
<table width="90%">
    <thead>
        <tr><td width="30%">名称</td><td width="30%">类型</td><td>创建日期
```

```
        </td></tr>
    </thead>
    <tbody id="tbody" runat="server"></tbody>
</table>
```

（2）在后台页面的 Load 事件中添加代码，调用 DirectoryInfo 类的 GetFileSystemInfos() 方法获取指定目录下的所有子目录和文件，然后进行遍历。代码如下。

```
DirectoryInfo di = new DirectoryInfo(Server.MapPath("~") + "Directory
Oper");
    if (di.Exists) {                              //如果查看的目录存在
        FileSystemInfo[] sysList = di.GetFileSystemInfos();
                                                  //获取目录的子目录和文件
        string info = "";
        foreach (FileSystemInfo item in sysList) {  //遍历获取到的目录
            if (item is DirectoryInfo)            //如果是文件夹
                info+="<tr><td>"+item.Name+"</td><td>文件夹</td><td>"+item.
                CreationTime+"</td>";
            else                                  //如果是文件
                info+="<tr><td>"+item.Name+"</td><td>"+item.Extension+"文
                件</td><td>"+item.CreationTime+"</td>";
    }
    tbody.InnerHtml = info;
}
else
    tbody.InnerText = "很抱歉，您要查看的目录并不存在。";
```

（3）运行页面查看效果，如图 12-3 所示。



图 12–3　遍历目录

提示

除了对目录进行创建、删除和遍历等操作外，还可以对目录进行移动、复制等操作，这里不再详细解释。

12.4　第三方控件

在介绍数据绑定控件时，通过内置的 PagedDataSource 类实现分页功能。除了这种

方法外，还可以使用其他的方法实现分页，例如第三方控件。

顾名思义，“第三方”就好比“第三者”，除了“我”和“你”之外的另外一方，“我”指用户本身，“你”指系统软件本身。另外一方相当于别的软件提供商。控件是用户可与之交互以输入或操作数据的对象，下面介绍常用的两个第三方控件。

12.4.1 分页控件

数据绑定控件实现数据分页时，可以通过内置的分页功能（如 GridView），可以利用 ASP.NET 提供的 DataPager 控件，但是 GridView 和 Repeater 都无法直接使用 DataPager 分页。最常用的一种分页方法就是利用第三方控件 AspNetPager，它将分页导航功能与数据显示功能完全独立开来，由用户自己控制数据的获取和显示方式，因此可以被灵活地应用于任何需要实现分页导航功能的地方。

1. 常用属性

AspNetPager 控件提供了多个属性，如表 12-5 所示列举了常用属性。

表 12-5 AspNetPager 控件的常用属性

属性名称	说明
AlwaysShow	获取或设置一个值，该值指定是否总是显示 AspNetPager 分页控件，即使要分页的数据只有一页
AlwayShowFirstLastPageNumber	获取或设置一个值，该值指定是否总显示第一页和最后一页的索引按钮
CssClass	获取或设置由 Web 服务器控件在客户端呈现的级联样式表（CSS）类
CurrentPageButtonPostion	当前页数字按钮在所有数字分页按钮中的位置，其值有 Fixed（默认固定）、Beginning（最前）、End（最后）和 Center（居中）
CurrentPageIndex	获取或设置当前显示页面的索引
Direction	获取或设置在 Panel 控件中显示包含文本的控件的方向
EnableUrlRewriting	获取或设置一个值，该值指定是否启用 URL 重写
FirstPageText	获取或设置为【第一页】按钮显示的文本
LastPageText	获取或设置为【最后一页】按钮显示的文本
NextPageText	获取或设置为【下一页】按扭显示的文本
PrevPageText	获取或设置为【上一页】按钮显示的文本
NavigationButtonsPosition	【首页】、【上页】、【下页】和【尾页】4 个导航按钮在分页导航元素中的位置，可选值为：Left（左侧）、Right（右侧）和 BothSides（默认值，分布在两侧）
NavigationButtonType	获取或设置【第一页】、【上一页】、【下一页】和【最后一页】按钮的类型，该值仅当 PagingButtonType 设为 Image 时才有效。其值有 Image 和 Text（默认值）
PageCount	获取所有要分页的记录需要的总页数
RecordCount	获取或设置需要分页的所有记录的总数
ShowFirstLast	获取或设置一个值，该值指示是否在页导航元素中显示第一页和最后一页按钮

续表

属性名称	说明
ShowMoreButtons	获取或设置一个值，该值指示是否在页导航元素中显示更多页按钮
CurrentPageIndex	获取或设置当前显示页的索引
ShowPageIndex	获取或设置一个值，该值指示是否在页导航中显示页索引数值按钮
ShowPageIndexBox	获取或设置页索引框的显示方式，以使用户输入或从下拉框中选择需要跳转到的页索引
CustomInfoHTML	获取或设置显示在用户自定义信息区的用户自定义 HTML 文本内容
SubmitButtonImageUrl	获取或设置【提交】按钮的图片路径，若该属性值为空则显示为普通按钮；否则显示为图片按钮且使用该属性的值作为图片路径

2．使用步骤

由于 AspNetPager 是第三方控件，因此在使用之前需要进行下载，然后将其拖动到工具箱中。步骤如下。

（1）在 AspNetPager 网站（http://www.webdiyer.com/Controls/AspNetPager/Downloads）下载最新版本的控件，下载完成后，解压缩包。

（2）在解压后的包中找到 AspNetPager.dll 文件，并将该文件引入到工具箱中。

（3）从工具箱中拖动此控件到页面的合适位置。

（4）设置与分页有关的内容。

【范例 16】

将 AspNetPager.dll 文件引入到工具箱时的步骤如下。

（1）打开工具箱并找到最后一个选项卡（根据需要也可以选择其他选项卡）后单击右键，如图 12-4 所示。

（2）单击图 12-4 中的【选择项】命令，弹出如图 12-5 所示的对话框。



图 12-4　工具箱

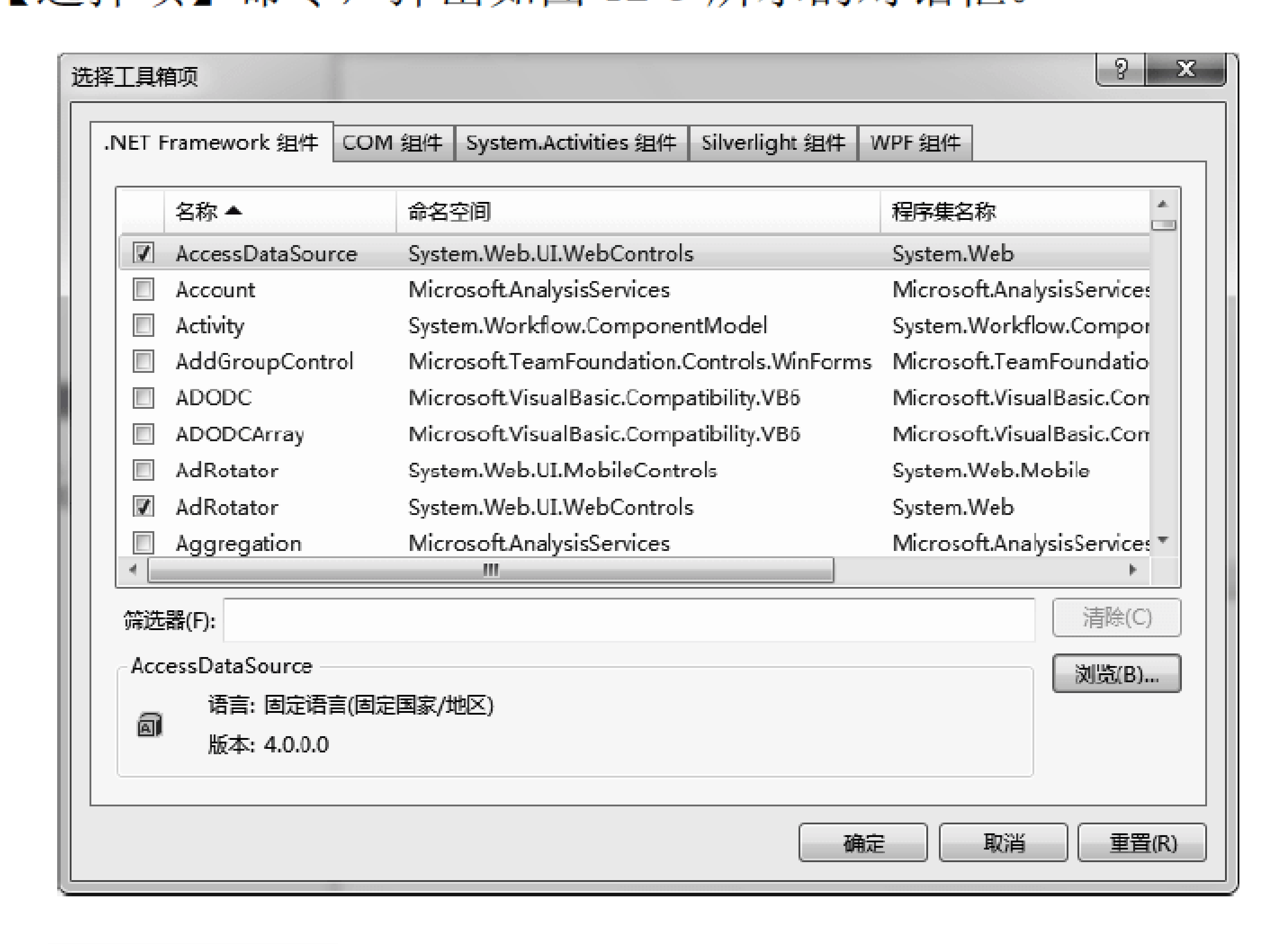


图 12-5　【选择工具箱项】对话框

（3）单击图 12-5 中的【浏览】按钮，在弹出的对话框中选择 AspNetPager.dll 文件，选择后进行添加。添加完毕后，会在【常规】选项卡中显示一个 AspNetPager 控件，效果图不再显示。

3. AspNetPager 控件的使用

将 AspNetPager 控件添加到工具箱之后就可以使用了，将其拖动到页面时会先注册该控件。有关代码如下。

```
<%@ Register Assembly="AspNetPager" Namespace="Wuqi.Webdiyer" TagPrefix=
"webdiyer" %>
<webdiyer:AspNetPager ID="AspNetPager1" runat="server"></webdiyer: Asp
NetPager>
```

【范例 17】

从 BookInfo 数据库表中读取全部记录，并利用 Repeater 控件与 AspNetPager 控件将数据分页显示。步骤如下。

（1）在页面中添加 Repeater 控件，在该控件的项模板中绑定数据，代码不再显示。

（2）在后台页面的 Load 事件中添加代码，首次加载时设置 AspNetPager 控件的 RecordCount 属性，并调用 Binding()方法绑定控件。代码如下。


```
protected void Page Load(object sender, EventArgs e) {
    if (!IsPostBack) {                        //页面首次加载
        AspNetPager1.RecordCount = Convert.ToInt32(SqlHelper. Execute
        Scalar(SqlHelper.connString, CommandType.Text, "SELECT COUNT(*)
        FROM BookInfo", null));               //总记录数
        Binding();
    }
}
```

（3）Binding()方法从数据库表中读取指定的数据，并将数据绑定到 Repeater 控件。代码如下。

```
public void Binding() {
    int page = AspNetPager1.PageSize;              //获取每页显示的条数
    int pageindex = (AspNetPager1.CurrentPageIndex - 1) * page;
    string sql = "SELECT TOP " + page + " * FROM BookInfo WHERE BookIsbn
     NOT IN (SEL " + pageindex + " BookIsbn FROM BookInfo) ";
    Repeater1.DataSource = SqlHelper.GetDataSet(SqlHelper.connString,
     CommandType.Text, sql, null);
    Repeater1.DataBind();
}
```

（4）为 AspNetPager 控件添加 PageChanged 事件，在该事件中调用 Binding()方法，代码不再显示。

（5）运行页面查看效果，分页效果如图 12-6 所示。



图 12-6 AspNetPager 控件的使用

12.4.2 验证码控件

ASP.NET 中并不提供生成验证码的控件，如果要生成验证码，需要借助于第三方的验证码控件或者第三方类。SerialNumber 控件是第三方验证控件，它包含两个常用方法。Create()方法用于自动生成验证码，CheckSN()方法返回一个布尔值，用于验证用户输入的验证码是否与自动生成的验证码一致。

SerialNumber 控件的使用与 AspNetPager 一致，使用前需要先下载一个 Webvalidates.dll 文件，然后将其添加到工具箱中。

注意 将 SerialNumber 控件拖动到工具箱时，页面【设计】窗口显示类似于“创建控件时出错”的提示，可以将其忽略。另外，验证控件内容时不区分用户输入的大小写，但是会区分全角和半角。

【范例 18】

直接使用 SerialNumber 控件生成验证码，并判断用户输入的验证码是否与生成的一致。步骤如下。

（1）在页面中添加多行两列的表格，模拟实现用户的登录。表格中包括登录名、密码、验证码和【登录】按钮，页面代码不再显示。

（2）将 SerialNumber 控件拖动到页面中，这时会自动注册该控件。有关代码如下。

```
<%@ Register Assembly="WebValidates" Namespace="WebValidates" TagPrefix=
"cc1" %>
<cc1:SerialNumber ID="SerialNumber1" runat="server"></cc1:SerialNumber>
```

（3）在后台页面添加代码，首次加载时调用 SerialNumber 控件的 Create()方法生成验证码，具体代码不再显示。

（4）为按钮添加 Click 事件，在事件代码中调用 CheckCode()方法判断用户输入的验证码是否正确，并弹出提示。代码如下。

```
protected void btnLogin Click(object sender, EventArgs e) {
    if (!CheckCode())        //如果验证码不正确，则显示提示信息
        Response.Write("<script>alert('验证码出错');</script>");
    else
        Response.Write("<script>alert('验证码正确');</script>");
}
```

（5）在 CheckCode()方法中调用 CheckSN()方法判断用户输入的验证码，代码如下。

```
protected bool CheckCode() {
    if (SerialNumber1.CheckSN(txtCode.Text.Trim()))
                                              //如果用户输入的验证码不正确
        return true;
    else {
        SerialNumber1.Create();               //重新生成验证码
        return false;
    }
}
```


（6）运行页面查看效果，生成的验证码如图 12-7 所示。在图 12-7 中输入内容进行测试，效果不再显示。

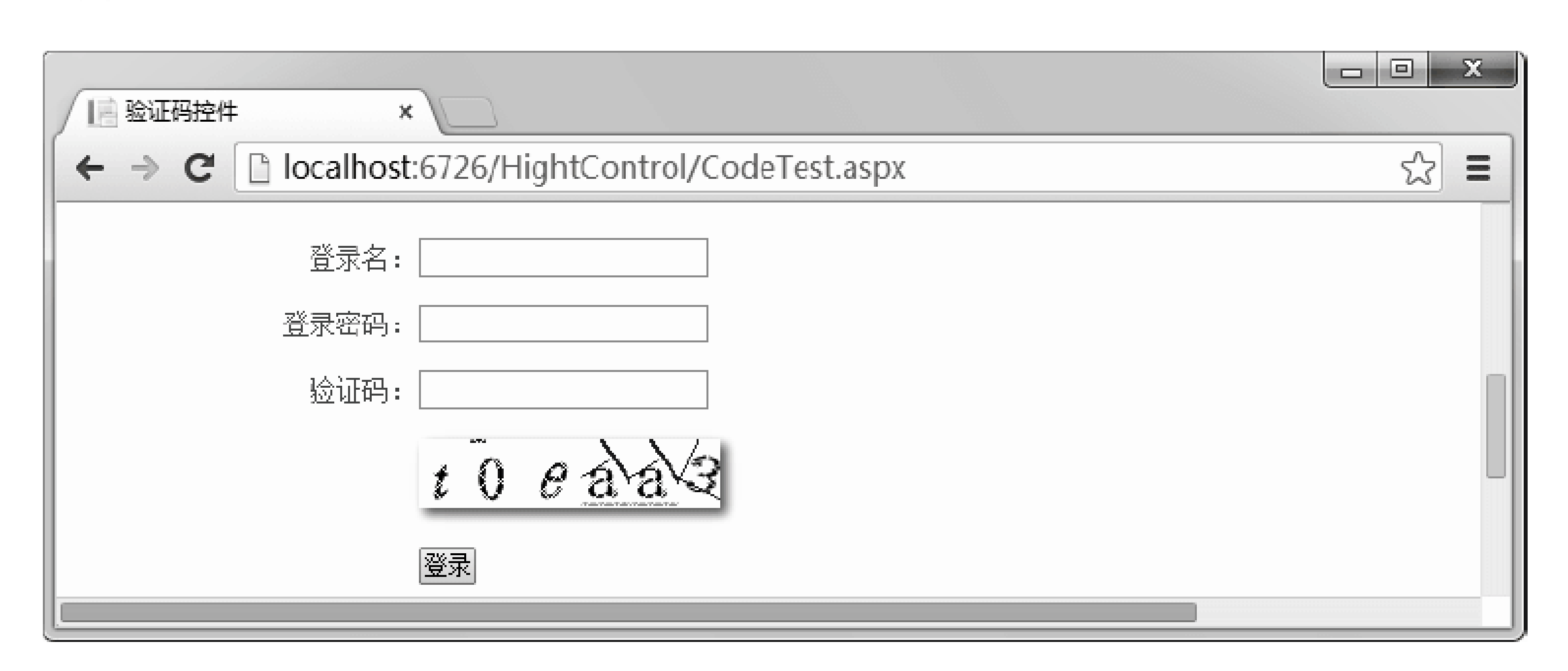


图 12-7　生成验证码效果

提示

使用 SerialNumber 控件生成验证码时的灵活度不高，安全性不高，并且不容易控制显示时的宽度和高度，这时还可以自定义验证码类生成验证码，并且对用户输入的全角和半角进行转换、对大小写进行转换等。

12.5　实验指导——WebSocket 发送消息

HTML 5 作为下一代 Web 标准，拥有许多引人注目的新特性，如 Canvas、本地存储、

多媒体编程接口以及 WebSocket 等。WebSocket 的出现是基于 Web 应用程序的实时性需要而产生的，这种实时的 Web 应用读者应该不陌生，如新浪微博的评论、私信的通知和腾讯的 WebQQ 等都用到了 WebSocket。

本节实验指导通过一个简单的例子演示如何通过 WebSocket 实现消息的发送。步骤如下。

（1）在当前的控制台应用程序中创建 index.html 页面，在页面中添加三个 button 元素和一个 span 元素，分别为 button 元素添加 Click 事件。代码如下。

```
<button id="ToggleConnection" type="button" onclick="ConnectionClicked(); ">
连接服务器</button>
<button id="Button1" type="button" onclick="SendData();">发消息：</button>
<span><br />昵称：小绵羊    年龄：15 岁   
 爱好：唱歌、跳舞和画画</span>
<button id="Button2" type="button" onclick="SeeState();">查看状态</button>
```

（2）在 JavaScript 脚本中添加 ConnectionClicked()函数，在该函数中为 ws 赋值并添加事件。代码如下。

```
var ws;
function ConnectionClicked() {
    try {
        ws =new WebSocket("ws://localhost:8080//WebSocket"); //连接服务器
        ws.onopen = function (event) { alert("已经与服务器建立了连接\r\n 当前
        连接状态：" + this.readyState); };
        ws.onmessage = function (event) { alert("接收到服务器发送的数据：\r\n"
        + event.data); };
        ws.onclose = function (event) { alert("已经与服务器断开连接\r\n 当前
        连接状态：" + this.readyState); };
        ws.onerror = function (event) {
            alert("WebSocket 异常！");
        };
    } catch (ex) {
        alert(ex.message);
    }
};
```

（3）添加 SendData()函数，在函数中调用 send()方法发送消息。代码如下。

```
function SendData() {
    try {
        var msg = { name: "小绵羊", age: "15 岁", love: "唱歌、跳舞和画画" };
        ws.send(JSON.stringify(msg));
    } catch (ex) {
        alert(ex.message);
    }
};
```

（4）添加 SeeState()函数，在函数中获取 ws 对象的 readyState 属性的值，并弹出

提示。

（5）在控制台应用程序中自动生成的 Program.cs 文件中添加代码，首先在 Main()方法中声明端口号和字节类型，然后再创建将网络端点表示为 IP 地址和端口号的 IPEndPoint 类的实例对象 localEP 和实现套接字接口的 Socket 实例对象 listener。代码如下。

```
static void Main(string[] args) {
    int port = 8080;
    byte[] buffer = new byte[1024];
    IPEndPoint localEP = new IPEndPoint(IPAddress.Any, port);
    Socket listener = new Socket(localEP.Address.AddressFamily, Socket
    Type.Stream, ProtocolType.Tcp);
    try {
        listener.Bind(localEP);
        listener.Listen(10);
        Console.WriteLine("等待客户端连接....");
        Socket sc = listener.Accept();          //接收一个连接
        Console.WriteLine("接收到了客户端：" + sc.RemoteEndPoint.ToString()
         + "连接....");
        //握手
        int length = sc.Receive(buffer);//接收客户端握手信息
        sc.Send(PackHandShakeData(GetSecKeyAccetp(buffer, length)));
        Console.WriteLine("已经发送握手协议了....");
        //接收客户端数据
        Console.WriteLine("等待客户端数据....");
        length = sc.Receive(buffer);//接收客户端信息
        string clientMsg = AnalyticData(buffer, length);
        Console.WriteLine("接收到客户端数据：" + clientMsg);
        //发送数据
        string sendMsg = "";
        string[] sendMsgInfo = clientMsg.Split(',');
        sendMsg += "我叫" + sendMsgInfo[0].Split(':')[1] + "，今年刚满 " +
       sendMsgInfo[1].Split(':')[1] + "，我的人生格言是：" + sendMsgInfo[2].
       Split(':')[1];
        Console.WriteLine("发送数据：“" + sendMsg + "” 至客户端....");
        sc.Send(PackData(sendMsg));
        Console.WriteLine("演示 Over!");
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

（6）根据需要添加其他代码，这里不再介绍这些代码。

（7）分别运行 Program.cs 文件和 index.html 页面，如图 12-8 所示。



图 12–8 运行文件和页面效果

（8）单击 index.html 页面中的【连接服务器】按钮进行测试，弹出对话框提示并在控制台中输入结果，如图 12-9 所示。



图 12–9 单击【连接服务器】按钮

（9）分别单击 index.html 页面中的【发消息】按钮和【查看状态】按钮进行测试，效果图不再显示。

思考与练习

一、填空题

1. ASP.NET 中提供的文件上传控件是__________。

2. __________类是一个静态类，因此可以直接调用其静态方法对文件进行操作。

3. FileInfo 类的__________属性获取表示文件扩展名部分的字符串。

4. 在下面代码中，横线处的内容是__________。

```
if(File.       (@"F:\test.doc"))
   lblResult.Text = "文件已经存在";
else
    lblResult.Text = "文件不存在";
```

二、选择题

1．实现文件下载时主要利用__________对象。

A．Session

B．Application

C．Response

D．Request

2．FileInfo 类的__________方法用于移动文件。

A．Move()

B．MoveTo()

C．Copy()

D．CopyTo()

3．获取目录信息时，DirectoryInfo 类的__________属性获取指定子目录的父目录名称。

A．Exists

B．Name

C．Parent

D．Root

4．DirectoryInfo 类的__________方法返回表示某个目录中所有文件和子目录的强类型 FileSystemInfo 项的数组。

A．GetDirectories()

B．GetFiles()

C．GetFileSystemInfos()

D．GetFileSystemEntries()

5．通过 AspNetPager 控件实现分页时，__________属性获取或设置需要分页的所有记录的总数。

A．PageSize

B．TotalCount

C．PageCount

D．RecordCount

三、简答题

1．实现文件下载时的一般步骤有哪些？

2．对文件进行创建、删除、复制和移动操作时需要调用哪些方法？

3．对目录进行创建、删除、移动和遍历操作时需要调用哪些方法？

4．第三方分页控件的常用属性有哪些？这些属性是做什么的？

第 13 章　Ajax 技术

Ajax 即 Asynchronous JavaScript and XML（异步 JavaScript 和 XML），是一种创建交互式网页应用的网页开发技术。Ajax 技术的重要应用是页面的异步更新。

传统的网页如果需要更新内容，必须重载整个网页；使用 Ajax 技术可以在不重新加载整个网页的情况下，对网页的某部分进行更新。本章详细介绍 Ajax 技术，包括 Ajax 技术的基础知识、内置的 Ajax 控件和 Ajax 技术的应用等。

本章学习要点：

- 了解 Ajax 技术标准
- 理解 Visual Studio 内置的 Ajax 控件
- 掌握 ScriptManager 控件的使用
- 掌握 Timer 控件的使用
- 掌握 UpdatePanel 控件的使用
- 掌握 UpdateProgress 控件的使用
- 理解 jQuery 编程技术
- 掌握 jQuery 的使用

13.1 Ajax 技术简介

Ajax 这个术语描述从基于 Web 的应用到基于数据的应用的转换。在基于数据的应用中，用户需求的数据可以从独立于实际网页的服务端取得，并且可以被动态地写入网页中。这样的数据转换速度，使得页面应用效果与桌面应用效果一样。

Ajax 的核心是 JavaScript 的 XmlHttpRequest 对象。该对象在 Internet Explorer 5 中首次引入，它是一种支持异步请求的技术。简而言之，XmlHttpRequest 可以使用 JavaScript 向服务器提出请求并处理响应，而不阻塞用户。

Ajax 不是一种新的编程语言，而是一种用于创建更好更快以及交互性更强的 Web 应用程序的技术。

Ajax 在浏览器与 Web 服务器之间使用异步数据传输（HTTP 请求），这样就可使网页从服务器请求少量的信息，而不是整个页面。

Ajax 基于下列 Web 标准：JavaScript、XML、HTML、CSS。在 Ajax 中使用的 Web 标准已被良好定义，并被所有的主流浏览器支持。Ajax 应用程序独立于浏览器和平台。

Ajax 技术在 1998 年前后得到了应用。允许客户端脚本发送 HTTP 请求（XMLHTTP）的第一个组件由 Outlook Web Access 小组写成。该组件原属于微软 Exchange Server，并且迅速地成为 Internet Explorer 4.0 的一部分。部分观察家认为，Outlook Web Access 是

第一个应用了 Ajax 技术的成功的商业应用程序，并成为包括 Oddpost 的网络邮件产品在内的许多产品的领头羊。但是，2005 年年初，许多事件使得 Ajax 被大众所接受。Google 在它著名的交互应用程序中使用了异步通信，如 Google、Google 地图、Google 搜索。

Ajax 前景非常乐观，可以提高系统性能，优化用户界面。Ajax 现有直接框架 AjaxPro，可以引入 AjaxPro.2.dll 文件，可以直接在前台页面调用后台页面的方法。但此框架与 FORM 验证有冲突。另外，微软也引入了 Ajax 组件，需要添加 AjaxControlToolkit.dll 文件，可以在控件列表中出现相关控件。

重要的技术和 Ajax 开发模式可以从现有的知识中获取。例如，在一个发送请求到服务端的应用中，必须包含请求顺序、优先级、超时响应、错误处理及回调，其中许多元素已经在 Web 服务中包含。同时，随着技术的成熟还会有许多地方需要改进，特别是 UI 部分的易用性。

Ajax 开发与传统的 C/S 开发有很大的不同。这些不同引入了新的编程问题，最大的问题在于易用性。由于 Ajax 依赖浏览器的 JavaScript 和 XML，浏览器的兼容性和支持的标准也变得和 JavaScript 的运行时性能一样重要了。这些问题中的大部分来源于浏览器、服务器和技术的组合，因此必须理解如何才能最好地使用这些技术。

综合各种变化的技术和强耦合的客户服务端环境，Ajax 提出了一种新的开发方式。Ajax 开发人员必须理解传统的 MVC 架构，这限制了应用层次之间的边界。同时，开发人员还需要考虑 C/S 环境的外部和使用 Ajax 技术来重定型 MVC 边界。最重要的是，Ajax 开发人员必须禁止以页面集合的方式来考虑 Web 应用而需要将其认为是单个页面。一旦 UI 设计与服务架构之间的范围被严格区分开来后，开发人员就需要更新和变化的技术集合了。

随着 Ajax 迅速地引人注目起来，开发人员对这种技术的期待也迅速地增加。就像任何新技术，Ajax 的兴旺也需要整个开发工具/编程语言及相关技术系统来支撑。

Visual Studio 中有着内置的几个 Ajax 控件：ScripManager、ScriptManagerProxy、Timer、UpdatePanel 和 UpdateProgress。

ScriptManager 与 UpdatePanel 这两个控件是 Visual Studio 中使用最多的控件之一，主要作用是对网页的内容进行局部的更新。

13.2 ScriptManager 控件

ScriptManager 控件是实现页面异步更新的基础。默认情况下，ScriptManager 控件会向页面注册 Microsoft Ajax Library 的脚本。这将使客户端脚本能够使用类型系统扩展，并支持部分页呈现和 Web 服务调用这样的功能。本节介绍 ScriptManager 控件的概念及其应用。

13.2.1 ScriptManager 简介

ASP.NET 的 Ajax 功能只有在页面使用 ScriptManager 控件的基础上才能够实现，

ScriptManager 控件有如下几个特点。

（1）它是脚本控制器，是 asp.net.Ajax 存在的基础，是所有的 Ajax 控件的基础。

（2）一个页面只允许有一个 ScriptManager，并且放在其他的 Ajax 控件的前面。

（3）ScriptManager 掌管着客户端 Ajax 页的多个脚本，并在页面中注册 Ajax 类库，用来实现页面的局部更新和对 Web 服务的调用。

Ajax 可向 JavaScript 添加类型系统扩展，以提供命名空间、继承、接口、枚举、映射以及字符串和数组的 Helper 函数。这些扩展可以在客户端脚本中提供与.NET Framework 的功能类似的功能。

利用这些扩展，可按一种结构化方式编写支持 Ajax 的 ASP.NET 应用程序，这不仅能提高可维护性，还使添加功能和功能分层的操作更容易。向 ASP.NET 网页添加 ScriptManager 控件自动包括类型系统扩展，可以在客户端脚本中使用该库。ScripManager 控件的基本属性如表 13-1 所示。

表 13-1 ScriptManager 控件的基本属性

属性名称	说明
AllowCustomError	是否要使用错误处理
AsyncPostBackErrorMessage	异步返回错误的时候是否返回错误信息
AsypostBackTimeOut	异步返回事件显示，默认为 90s
EnablePartialRendering	是否支持页面的局部刷新
ScriptMode	指定发送到客户端的脚本模式，有 4 种取值：Auto、Inherit、Debug、Release，默认为 Auto
ScriptPath	设置所有的脚本块的根目录，作为全局属性

虽然 Ajax 功能需要借助 ScriptManager 控件才能够起作用，但 ScriptManager 控件是不在页面中显示的，而是控制着页面中的 UpdatePanel 控件来显示。这样的关系相当于数据源控件与数据显示控件，数据源控件能够获取数据但不显示，而数据显示控件能够显示数据源中的数据。当页包含一个或多个 UpdatePanel 控件时，ScriptManager 控件将管理浏览器中的部分页呈现，与页生命周期进行交互，更新 UpdatePanel 控件内的区域。

13.2.2 ScriptManager 应用

ScriptManager 控件是 ASP.NET 中 Ajax 功能的中心。ScriptManager 控件管理一个页面上的所有 Ajax 资源。其中包括将 Ajax 脚本下载到浏览器和协调通过使用 UpdatePanel 控件启用的部分页面更新。此外，通过 ScriptManager 控件还能执行以下操作。

（1）注册与部分页面更新兼容的脚本。为了管理脚本与核心库之间的依赖项，将在加载 Ajax 脚本之后加载注册的所有脚本。

（2）指定是发布还是调试发送到浏览器的脚本。

（3）通过向 ScriptManager 控件注册 Web 服务，提供从脚本访问 Web 服务方法的权限。

（4）通过向 ScriptManager 控件注册 ASP.NET 身份验证、角色和配置文件应用程序服务，提供从客户端脚本访问这些服务的权限。

（5）在浏览器中以区域性特定的形式显示脚本的 Date、Number 和 String 函数。

（6）使用 ScriptReference 控件的 ResourceUICultures 属性来访问嵌入式脚本文件或独立脚本文件的本地化资源。

（7）向 ScriptManager 控件注册可实现 IExtenderControl 或 IScriptControl 接口的服务器控件，以便呈现客户端组件和行为所需的脚本。

一个网页只能包含一个 ScriptManager 控件，该控件既可以直接位于页面中，也可以间接位于嵌套组件或父组件内部。下面详细介绍 ScriptManager 控件的应用。

1．脚本的管理和注册

ScriptManager 控件控制脚本的使用，可以直接通过控件的集合，也可通过注册方法，再指定脚本。

ScriptManager 控件的 Scripts 集合中针对浏览器中可用的每个脚本包含一个 ScriptReference 对象。可以以声明方式或编程方式指定脚本。

ScriptManager 控件公开了一些注册方法，以编程方式管理客户端脚本和隐藏字段。当为支持部分页更新的脚本或隐藏字段注册时，必须调用 ScriptManager 控件的注册方法。

页上使用 ScriptManager 控件注册的脚本以及所有事件处理脚本，必须位于页上的 form 元素内。否则，将不会注册或执行脚本。

通过 ScriptManager 控件可注册随后将作为页面一部分呈现的脚本。ScriptManager 控件注册方法可以细分为以下三种类别。

（1）保证维护 Ajax 上脚本依赖项的注册方法。

（2）不依赖 Ajax 但与 UpdatePanel 控件兼容的注册方法。

（3）支持与 UpdatePanel 控件协作的注册方法。

注册依赖 Ajax 的脚本可以使用表 13-2 中的方法，以保证维护 Ajax 上所有依赖项的方式注册脚本文件。

表 13-2　维护 Ajax 上所有依赖项的注册方法

方法名称	说明
RegisterScriptControl<TScriptControl>	注册可实现用来定义 Sys.Component 客户端对象的 IScriptControl 接口的服务器控件。ScriptManager 控件呈现支持该客户端对象的脚本
RegisterExtenderControl<TExtenderControl>	注册可实现用来定义 Sys.UI.Behavior 客户端对象的 IExtenderControl 接口的服务器控件。ScriptManager 控件呈现支持该客户端对象的脚本

可以使用表 13-3 中的方法注册不依赖 Ajax 但与 UpdatePanel 控件兼容的脚本文件。这些方法与 ClientScriptManager 控件的类似方法相对应。如果为便于在 UpdatePanel 控件中使用而呈现脚本，则应确保调用 ScriptManager 控件的方法。

表13-3 与 UpdatePanel 控件兼容的脚本的注册方法

方法名称	说明
RegisterArrayDeclaration()	在 JavaScript 数组中添加值。如果该数组不存在，则创建它
RegisterClientScriptBlock()	在页面的<form>开始标记之后呈现一个 script 元素。该脚本被指定为字符串参数
RegisterClientScriptInclude()	在页面的<form>开始标记之后呈现一个 script 元素。通过将 src 特性设置为指向脚本文件的 URL 来指定脚本内容
RegisterClientScriptResource()	在页面的<form>开始标记之后呈现一个 script 元素。脚本内容是使用程序集中的资源名称指定的。通过调用从程序集中检索命名脚本的 HTTP 处理程序，来使用 URL 自动填充 src 特性
RegisterExpandoAttribute()	在标记中为指定控件呈现一个自定义名称/值特性对（一个 expando）
RegisterHiddenField()	呈现隐藏字段
RegisterOnSubmitStatement()	注册为响应 form 元素的 submit 事件而执行的脚本。onSubmit 特性引用指定脚本
RegisterStartupScript()	在页面的</form>结束标记之前呈现启动脚本块。要呈现的脚本被指定为字符串参数

在注册方法时，可为该脚本指定类型/键对。如果已注册了一个包含相同类型/键对的脚本，则不会注册新的脚本。同样，如果所注册脚本的类型/资源名称对已存在，则不会再添加引用该资源的 script 元素。如果注册一个以前注册过的 expando 特性，则会引发异常。允许重复注册数组值。

在调用 RegisterClientScriptInclude()或 RegisterClientScriptResource()方法时，应避免注册执行内联函数的脚本。相反，应注册包含函数定义（如事件处理程序）或应用程序的自定义类定义的脚本。

2. Web 服务应用

若要注册想要从支持 Ajax 的 ASP.NET 页调用的 Web 服务，可以通过将该 Web 服务添加到 ScriptManager 控件的 Services 集合来注册它。ASP.NET Ajax Framework 为 Services 集合中的每个 ServiceReference 对象生成一个客户端代理对象。这些代理类及其强类型成员将简化从客户端脚本使用 Web 服务的过程。

可以以编程方式将 ServiceReference 对象添加到 Services 集合，以便在运行时为 Web 服务注册。

通过创建一个 ServiceReference 对象，然后将其添加到 ScriptManager 控件的 Services 集合中，可以注册一个要从客户端脚本调用的 Web 服务。ASP.NET 可为 Services 集合中的每个 ServiceReference 对象生成一个客户端代理对象。可通过编程方式将 ServiceReference 对象添加到 Services 集合中，以便在运行时注册 Web 服务。

ScriptManager 控件可在呈现页面中生成指向适当的本地化脚本文件（嵌入程序集中的脚本文件或独立脚本文件）的引用。

在将 EnableScriptLocalization 属性设置为 true 时，ScriptManager 控件会检索当前区域中诸如本地化字符串这样的本地化资源（如果存在）。ScriptManager 控件可为使用本地化资源提供下列功能。

1）嵌入到程序集中的脚本文件

ScriptManager 控件可确定将哪个区域性特定的或回退区域性脚本文件发送到浏览器。为此，它会使用区域性特定的 NeutralResourcesLanguageAttribute 程序集特性、打包在程序集中的资源以及浏览器的 UI 区域性（如果有）。

2）独立脚本文件

ScriptManager 控件可使用 ScriptReference 对象的 ResourceUICultures 属性来定义受支持的 UI 区域性的列表。

3）在调试模式中

ScriptManager 控件尝试呈现包含调试信息的区域性特定的脚本文件。例如，如果页面处于调试模式且当前区域性设置为 en-MX，则该控件会呈现一个其名称如 scriptname.en-MX.debug.js 这样的脚本文件（如果该文件存在）。如果该文件不存在，则呈现适当回退区域性的调试文件。

3. 从客户端脚本使用身份验证、配置文件和角色服务

Ajax 包含用于从 JavaScript 直接调用 ASP.NET 2.0 Forms 身份验证、配置文件和角色应用程序服务的代理类。如果要使用自定义身份验证服务，则可通过使用 ScriptManager 控件来为该服务注册。

4. 添加特定于嵌套的组件的脚本和服务

只能向页添加 ScriptManager 控件的一个实例。该页可以直接包含该控件，也可以将其间接包含在嵌套的组件中，如用户控件、母版页的内容页或嵌套的母版页。如果页已包含 ScriptManager 控件，但嵌套的组件或父组件需要 ScriptManager 控件的其他功能，则该组件可以包含 ScriptManagerProxy 控件。

使用 ScriptManagerProxy 控件，可在母版页或宿主页已包含 ScriptManager 控件的情况下，将脚本和服务添加到内容页和用户控件中。

在使用 ScriptManagerProxy 控件时，可以将脚本和服务添加到 ScriptManager 控件所定义的脚本和服务集合。如果不希望在包含特定 ScriptManager 控件的每一页上都包含特定的脚本和服务，则可以将这些脚本和服务从 ScriptManager 控件中移除。可以通过改用 ScriptManagerProxy 控件，将这些脚本和服务添加到各页中。

5. 部分页面呈现

ScriptManager 控件的 EnablePartialRendering 属性确定某个页是否参与部分页更新。默认情况下，EnablePartialRendering 属性为 true。因此，默认情况下，当向页添加 ScriptManager 控件时，将启用部分页呈现。

ASP.NET 页面支持部分页面呈现的能力受到以下因素的控制。

（1）ScriptManager 控件的 EnablePartialRendering 属性必须为 true（默认值）。

（2）页面上必须至少有一个 UpdatePanel 控件。

（3）SupportsPartialRendering 属性必须为 true（默认值）。如果没有显式设置 SupportsPartialRendering 属性，则其值依浏览器功能而定。

可以在发生页面的 Init 事件期间或之前，在运行时重写 EnablePartialRendering 属性的值。如果尝试在发生页面的 Init 事件后更改此属性，则会引发 InvalidOperationException 异常。

当部分页面呈现受支持时，ScriptManager 控件会呈现脚本以启用异步回发和部分页面更新。可使用 UpdatePanel 控件来指定要更新的页面区域。ScriptManager 控件会处理异步回发，并且只刷新必须要更新的页面区域。

通常将 ScriptManager 控件与母版页、用户控件及其他子组件一起使用。

一个页面在其层次结构中只能包含一个 ScriptManager 控件。若要在父页面已具有 ScriptManager 控件时为嵌套页面、用户控件或组件注册服务和脚本，则需要使用 ScriptManagerProxy 控件。

13.3 UpdatePanel 控件

UpdatePanel 控件是 ASP.NET 中 Ajax 功能的中心部分。它们与 ScriptManager 控件一起使用，以启用部分页呈现。本节介绍 UpdatePanel 控件的概念及其应用。

13.3.1 UpdatePanel 简介

UpdatePanel 在 ScriptManager 控件开启了 Ajax 功能之后，为页面中需要局部更新的地方设置一个区域。UpdatePanel 控件相当于一个容器，在页面中占据一定的区域，该区域的内容可在页面整体没有更新的状态下进行更新，即局部更新。

UpdatePanel 控件通过指定页中无须刷新整个页面即可更新的区域发挥作用。此过程由 ScriptManager 服务器控件和客户端 PageRequestManager 类来协调。当启用部分页更新时，控件可以通过异步方式发布到服务器。异步回发的行为与常规回发类似：生成的服务器页执行完整的页和控件生命周期。不过，通过使用异步回发，可将页更新限制为包含在 UpdatePanel 控件中并标记为要更新的页区域。服务器仅将受影响的元素的 HTML 标记发送到浏览器。在浏览器中，客户端 PageRequestManager 类执行文档对象模型（DOM）操作以将现有 HTML 替换为更新的标记。

使用异步回发或使用 XMLHTTPRequest 对象时，如果 URL 包含双字节字符，则可能发生回发错误。此问题可以通过下面的方法得到解决：在页面的 head 元素中加入<base href="url"/>元素，其中 href 属性设置为引用该页面的 URL 编码的字符串。可以动态添加到服务器代码中的此元素。

UpdatePanel 控件的呈现在只需更新部分区域时减少了同步回发和更新整个页面的需要。由于部分区域更新减少了整页回发时的屏幕闪烁并提高了网页交互性，因而改善了用户体验。UpdatePanel 控件的常用属性如表 13-4 所示。

表 13-4　UpdatePanel 控件的常用属性

属性名称	说明
Attributes	获取 UpdatePanel 控件的级联样式表（CSS）特性集合
BindingContainer	基础结构。获取包含此控件数据绑定的控件
ChildControlsCreated	获取一个值服务器控件的子控件是否已创建
ChildrenAsTriggers	获取或设置一个值，该值指示来自 UpdatePanel 控件的即时子控件的回发是否更新面板的内容
ClientID	获取由 ASP.NET 生成的 HTML 标记的控件 ID
ContentTemplate	获取或设置定义 UpdatePanel 控件内容的模板
ContentTemplateContainer	获取一个可以以编程方式向其添加子控件的控件对象
Context	获取 HttpContext 对象与当前 Web 请求的服务器控件
EnableTheming	获取或设置一个主题淡出适用于此控件
IsChildControlStateCleared	获取指示该控件中包含的控件的值是否具有状态
IsInPartialRendering	获取一个值，该值指示是否因异步回发而更新 UpdatePanel 控件
Parent	具有引用页面控件层次结构的服务器控件的父控件
RenderingCompatibility	获取指定 ASP.NET 版本中 HTML 将兼容的值
RenderMode	获取或设置一个值，该值指示 UpdatePanel 控件的内容是否包含在 HTML<div>或<span>元素中
SkinID	获取或设置外观应用于控件
Triggers	获取一个 UpdatePanelTriggerCollection 对象，该对象包含以声明方式为 UpdatePanel 控件注册的 AsyncPostBackTrigger 和 PostBackTrigger 对象
UpdateMode	获取或设置一个值，该值指示何时更新 UpdatePanel 控件的内容

13.3.2　UpdatePanel 异步更新

启用 UpdatePanel 呈现后，控件可执行一个回发来更新整个页，也可执行异步回发来更新一个或多个 UpdatePanel 控件的内容。控件是否导致异步回发并更新 UpdatePanel 控件视以下情况而定，可根据 UpdatePanel 控件的 UpdateMode 属性来设置，如表 13-5 所示。

表 13-5　UpdateMode 属性与对应的页面更新情况

UpdateMode 属性	更新情况
Always	UpdatePanel 控件的内容在每次从页上的任意位置执行回发时都会更新。这包括来自其他 UpdatePanel 控件的异步回发，也包括来自 UpdatePanel 控件外部的控件的回发
Conditional	当使用 Triggers 属性定义为触发器的 UpdatePanel 控件导致回发时更新。在这种情况下，该控件显式触发面板内容的更新。该控件可以位于与触发器关联的 UpdatePanel 控件的内部或外部。 当显式调用 UpdatePanel 控件的 Update 方法时更新。 当 UpdatePanel 控件嵌套在另一个 UpdatePanel 控件内并更新父面板时更新。 当 ChildrenAsTriggers 属性设置为 true 并且 UpdatePanel 控件的子控件导致回发时更新。嵌套的 UpdatePanel 控件的子控件不会导致更新外部 UpdatePanel 控件，除非将它们显式定义为父面板的触发器

在 UpdateMode 设置为 Always 时不允许同时将 ChildrenAsTriggers 属性设置为 false，

这会引发异常。

当 UpdatePanel 控件执行异步发布时，会添加自定义 HTTP 头。一些代理会移除此自定义 HTTP 头。如果发生此情况，则服务器会将请求作为常规回发进行处理，这会引发客户端错误。若要解决此问题，需要在执行异步发布时插入自定义窗体字段。然后检查服务器代码中的头或自定义窗体字段。

可以使用多个 UpdatePanel 控件来单独更新不同的页区域。第一次呈现包含一个或多个 UpdatePanel 控件的页后，会呈现所有 UpdatePanel 控件的所有内容，并会将这些内容发送到浏览器。在随后执行异步回发时，根据面板设置以及各面板的客户端或服务器逻辑的不同，可能不会更新所有 UpdatePanel 控件的内容。

此外，还可以在以下情况下使用 UpdatePanel 控件。

（1）在用户控件中。

（2）在母版页和内容页上。

（3）嵌套在其他 UpdatePanel 控件中。

（4）在模板化控件（如 GridView 或 Repeater 控件）内部。

在使用 UpdatePanel 控件时，可使用表 13-6 中的方法来实现页面局部更新。

表 13-6 实现页面局部更新的方法

方法名称	说明
RegisterAsyncPostBackControl()	将控件注册为异步回发的触发器
RegisterDataItem()	在部分页面呈现期间将自定义数据发送到控件
RegisterDispose()	为 UpdatePanel 控件内的某个控件注册一个 dispose 脚本。在更新或删除 UpdatePanel 控件时会执行该脚本
RegisterPostBackControl()	将控件注册为完全回发的触发器。该方法用于 UpdatePanel 控件内以其他方式执行异步回发的控件

如果在异步回发期间出现页面错误，则会引发 AsyncPostBackError 事件。以何种方式将服务器上的错误发送到客户端取决于 AllowCustomErrorsRedirect 属性、AsyncPostBackErrorMessage 属性以及 Web.config 文件的自定义错误部分。

可以以编程方式添加 UpdatePanel 控件，但不能以编程方式添加触发器。若要创建类似触发器的行为，可以将页上的控件注册为异步回发控件。可以通过调用 ScriptManager 控件的 RegisterAsyncPostBackControl()方法来执行此操作。然后，可以创建一个为响应异步回发而运行的事件处理程序，并在该处理程序中调用 UpdatePanel 控件的 Update()方法。

可以通过声明方式向 UpdatePanel 控件添加内容，也可以在设计器中通过使用 ContentTemplate 属性来添加内容。在标记中，将此属性作为 ContentTemplate 元素公开。若要以编程方式添加内容，建议使用 ContentTemplateContainer 属性。

当首次呈现包含一个或多个 UpdatePanel 控件的页时，将呈现 UpdatePanel 控件的所有内容并将这些内容发送到浏览器。在后续异步回发中，可能会更新各个 UpdatePanel 控件的内容。更新将与面板设置、导致回发的元素以及特定于每个面板的代码有关。

默认情况下，UpdatePanel 控件内的任何回发控件都将导致异步回发并刷新面板的内容。但是，也可以配置页上的其他控件来刷新 UpdatePanel 控件。可以通过为 UpdatePanel

控件定义触发器来做到这一点。触发器是一类绑定，用于指定使面板更新的回发控件和事件。当引发触发器控件的指定事件（例如，按钮的 Click 事件）时，将刷新更新面板。

使用 UpdatePanel 控件的 Triggers 元素内的 asp:AsyncPostBackTrigger 元素定义触发器。如果在 Visual Studio 中编辑页面，则可以使用【UpdatePanelTrigger 集合编辑器】对话框创建触发器。

触发器的控件事件是可选的。如果不指定事件，则触发器事件是控件的默认事件。例如，对于 Button 控件，默认事件是 Click 事件。

【范例 1】

向页面中添加一个 ScriptManager 和一个 UpdatePanel 控件，分别在 UpdatePanel 控件内部和 UpdatePanel 控件外部添加一个按钮，在 UpdatePanel 控件外部添加一个标签，并定义页面的加载事件，判断页面是否是第一次加载。

（1）若页面是第一次加载，则标签的标题是“页面首次加载”。

（2）若页面不是第一次加载，则标签的标题是“感谢再次光临！”。

省略控件的添加步骤，页面加载事件代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)          //页面是否首次加载
    {
        num.Text = "页面首次加载";
    }
    else                           //页面非首次加载
    {
        num.Text = "感谢再次光临！";
    }
}
```

运行该页面，单击 UpdatePanel 控件内的按钮，页面没有变化，标签的标题是“页面首次加载”；而单击 UpdatePanel 控件外的按钮，标签的标题被修改为“感谢再次光临！”。可见 UpdatePanel 控件内的按钮只是进行了局部的更新，而 UpdatePanel 控件外的按钮可进行整页的更新。

13.3.3 异步回发中的应用限制

并不是所有的控件、方法和属性都支持页面的异步更新，其应用限制主要表现在三个方面：不支持异步更新的 ASP.NET 控件、不支持异步更新的 Web 控件、不支持异步更新的属性和方法。以下分别介绍不支持异步更新的控件、方法和属性。

1. 不支持异步更新的 ASP.NET 控件

下面的 ASP.NET 控件与部分页更新不兼容，因此不应在 UpdatePanel 控件内使用。

（1）处于多种情况下的 TreeView 控件。一种情况是启用了不是异步回发的一部分的

回调。另一种情况是将样式直接设置为控件属性，而不是通过使用对 CSS 样式的引用来隐式设置控件的样式。还有一种情况是 EnableClientScript 属性为 false（默认值为 true）。另外，还有一种情况是在异步回发之间更改了 EnableClientScript 属性的值。

（2）Menu 控件（将样式直接设置为控件属性，而不是通过使用对 CSS 样式的引用来隐式设置控件的样式时）。

（3）FileUpload 和 HtmlInputFile 控件（当它们作为异步回发的一部分用于上载文件时）。

（4）GridView 和 DetailsView 控件（当它们的 EnableSortingAndPagingCallbacks 属性设置为 true 时）。默认值为 false。

（5）Login、PasswordRecovery、ChangePassword 和 CreateUserWizard 控件（其内容尚未转换为可编辑的模板）。

（6）Substitution 控件。

若要在 UpdatePanel 控件内使用 FileUpload 或 HtmlInputFile 控件，需要将提交文件的回发控件设置为面板的 PostBackTrigger 控件。仅可以在回发方案中使用 FileUpload 和 HtmlInputFile 控件。

所有其他控件都可以在 UpdatePanel 控件内发挥作用。不过在某些情况下，控件在 UpdatePanel 控件内可能不会按预期方式工作。这些情况包括：

（1）通过调用 ClientScriptManager 控件的注册方法注册脚本。

（2）在该控件呈现过程中直接呈现脚本或标记，例如，通过调用 Write()方法。

如果该控件调用 ClientScriptManager 控件的脚本注册方法，则也许能够改用 ScriptManager 控件相应的脚本注册方法。在此情况下，该控件可以在 UpdatePanel 控件内工作。

2. 不支持异步更新的 Web 控件

Web 控件是一组集成控件，用于创建网站使最终用户可以直接从浏览器修改网页的内容、外观和行为。可以在 UpdatePanel 控件内使用 Web 控件，但具有以下限制。

（1）每个 WebPartZone 控件都必须在同一 UpdatePanel 控件内。例如，页上不能具有两个带有各自的 WebPartZone 控件的 UpdatePanel 控件。

（2）WebPartManager 控件管理 Web 部件控件的所有客户端状态信息。它必须在页面上的最外部的 UpdatePanel 控件内。

（3）不能通过使用异步回发导入或导出 Web 控件(执行此任务需要 FileUpload 控件，该控件不能与异步回发一起使用）。默认情况下，导入 Web 控件将执行完全回发。

（4）在异步回发过程中，不能添加或修改 Web 部件控件的样式。

3. 不支持异步更新的属性和方法

在异步回发过程中，不支持在页面上设置默认回发按钮的以下方案。

（1）在异步回发过程中以编程方式更改 DefaultButton。

（2）在异步回发过程中，当 Panel 控件不在 UpdatePanel 控件内部时以编程方式更改

DefaultButton。

仅在回发方案（不包括异步回发方案）下才支持 HttpResponse 类的以下方法：BinaryWrite()、Clear()、ClearContent()、ClearHeaders()、Close()、End()、Flush()、TransmitFile()、Write()、WriteFile()和 WriteSubstitution()。

13.3.4 UpdateProgress

使用 UpdateProgress 控件可以提供有关部分页呈现进度的反馈。对于回发或初始页呈现，不显示 UpdateProgress 控件内容。

页面可以包含多个 UpdateProgress 控件。每个控件都可以与不同的 UpdatePanel 控件关联。此外，还可以使用一个 UpdateProgress 控件，并将它与页上的所有 UpdatePanel 控件关联。

UpdateProgress 控件根据回发所源于的位置以及是否设置 UpdateProgress 控件的 AssociatedUpdatePanelID 属性，来呈现显示或隐藏的 div 元素。

通过设置 UpdateProgress 控件的 AssociatedUpdatePanelID 属性，将 UpdateProgress 控件与 UpdatePanel 控件关联。当回发事件源于 UpdatePanel 控件内部时，会显示所有关联的 UpdateProgress 控件。如果没有设置 AssociatedUpdatePanelID 属性，则 UpdateProgress 控件会为源于任何 UpdatePanel 控件内部的任何异步回发显示进度。还会为充当面板触发器的任何控件显示进度。

AssociatedUpdatePanelID 属性对 UpdateProgress 控件行为具有以下影响。

（1）如果没有设置 AssociatedUpdatePanelID 属性，则为以下回发显示 UpdateProgress 控件：源于任何 UpdatePanel 控件内部的回发；源于充当任何 UpdatePanel 控件的异步触发器的控件的回发。

（2）如果将 AssociatedUpdatePanelID 属性设置为 UpdatePanel 控件 ID，则会为源于关联 UpdatePanel 控件内部的回发显示 UpdateProgress 控件。

（3）如果将 AssociatedUpdatePanelID 属性设置为不存在的控件，则永远不会显示 UpdateProgress 控件。

（4）如果将 UpdatePanel 控件的 ChildrenAsTriggers 属性设置为 false，而回发源于 UpdatePanel 控件内部，则仍会显示任何关联的 UpdateProgress 控件。

使用 ProgressTemplate 属性可以指定由 UpdateProgress 控件显示的消息。如果 ProgressTemplate 属性为空，则在显示 UpdateProgress 控件时不会显示任何内容。

UpdateProgress 控件可以位于其他 UpdatePanel 控件的内部或外部。UpdateProgress 控件模板的显示与 UpdateProgress 控件所在的位置无关。在嵌套的 UpdatePanel 控件中，子面板位于父面板之内。在此情况下，源于子面板内部的回发会导致显示出与子面板和父面板关联的所有 UpdateProgress 控件。如果回发源于父面板的即时子控件，则只显示与父面板关联的 UpdateProgress 控件。

默认情况下，UpdateProgress 控件会在显示其内容之前等待 0.5s（500ms）。在异步回发的速度很快时，这有助于防止控件闪烁。通过设置 DisplayAfter 属性可指定延迟。

如果需要在显示 UpdateProgress 控件时进行更精细的控制，则可为 PageRequestManager 类的 beginRequest 和 endRequest 事件提供客户端脚本。在 beginRequest 事件处理程序中，可显示表示 UpdateProgress 控件的 DOM 元素。在 endRequest 事件处理程序中，可隐藏该元素。

在下列情况下，必须提供客户端脚本才能在更新目标 UpdatePanel 控件时显示 UpdateProgress 控件。

（1）将来自控件的回发注册为面板的异步回发触发器，并且页面上存在 UpdateProgress 控件。但是，AssociatedUpdatePanelID 属性未设置为面板的 ID。

（2）通过使用 ScriptManager 控件的 RegisterAsyncPostBackControl 方法将来自控件的回发注册为异步回发控件。

13.4 Timer 控件

使用 Timer 控件可以以指定间隔执行回发。在将 Timer 控件用作 UpdatePanel 控件的触发器时，使用异步更新或部分页更新来更新 UpdatePanel 控件。必须在网页中包含 ScriptManager 对象，才能使用 Timer 控件。

通过在 UpdatePanel 控件内部包含计时器，可使用 Timer 控件更新 UpdatePanel 控件。也可以将计时器放在 UpdatePanel 控件外部，然后将计时器设置为触发器。

此外，还可以将 Timer 控件包含在网页中并且不将其设置为 UpdatePanel 控件的触发器，以启动完整网页的完全回发。

可以为 Tick 事件创建事件处理程序，以在经过计时器间隔后运行服务器代码。在该事件处理程序中，可以包含代码来动态调整 Timer 控件的行为。

设置 Interval 属性可以指定执行回发的频率。设置 Enabled 属性可以启用或禁用 Timer。

Timer 控件回发到 Web 服务器时的准确性取决于正在浏览器中运行的 ECMAScript（JavaScript）window.setTimeout 函数的准确性。

Interval 属性是以 ms 为单位定义的。必须将 Interval 属性设置为特定值，以允许至少完成一次异步回发后才启动下次回发。如果 Timer 控件位于 UpdatePanel 控件外部，则在刷新 UpdatePanel 控件的内容时计时器会继续运行。如果在处理早期回发期间启动了新的回发，则会取消第一个回发。默认值是 60 000ms（60s）。

添加一个具有较小 Interval 属性值的 Timer 控件时，可能会与 Web 服务器之间产生大量的通信。使用 Timer 控件可以仅按所需的频率刷新内容。

Timer 控件可以与 UpdatePanel 控件结合使用，以定时更新 UpdatePanel 控件内的内容。

13.5 实验指导——图片的定时切换

根据本章的内容，使用 UpdatePanel 控件、ScriptManager 控件和 Timer 控件，实现页面在一定区域内的局部更新，定时切换图片，步骤如下。

（1）向页面中添加判断页面是否首次加载的标签 num，添加 UpdatePanel 控件和 ScriptManager 控件，在 UpdatePanel 控件中添加一个时间标签 Label1、一个图片控件和一个 Timer 控件，设置切换时间间隔为 1s，代码如下。

```
<asp:Label ID="num" runat="server" Text="" ForeColor="#9999FF"> </asp:
Label>
<asp:ScriptManager ID="ScriptManager1" runat="server"></asp:Script Manager>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:Label ID="Label1" runat="server" Text="当前时间"></asp:
        Label><br />
        <asp:Image ID="Image1" runat="server" ImageUrl="~/Images/ hill.
        jpg" />
        <asp:Timer ID="Timer1" runat="server" Interval="1000" OnTick=
        "Timer1 Tick"></asp:Timer>
    </ContentTemplate>
</asp:UpdatePanel>
```

（2）在页面加载时判断页面是否是首次加载，同时将时间标签的标题设置为当前时间，代码如下。


```
protected void Page Load(object sender, EventArgs e)
{
    Label1.Text = "当前时间：" + DateTime.Now.ToString();
    if (!Page.IsPostBack)        //页面是否首次加载
    {
         num.Text = "页面首次加载";
    }
    else                         //页面非首次加载
    {
        num.Text = "感谢再次光临！";
    }
}
```

（3）定义 Timer 控件的间隔回发事件，在指定的 1s 时间间隔后更新当前时间，同时切换图片控件的图片地址，代码如下。

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    Label1.Text = "当前时间：" + DateTime.Now.ToString();
    if (Image1.ImageUrl == "~/Images/hill.jpg")
    {
        Image1.ImageUrl = "~/Images/lake.jpg";
    }
    else
    { Image1.ImageUrl = "~/Images/hill.jpg"; }
}
```

（4）运行上述代码，可看到当前时间在不断变化着，同时图片也在不断切换着。如

在 18:11:35 时刻，页面效果如图 13-1 所示。

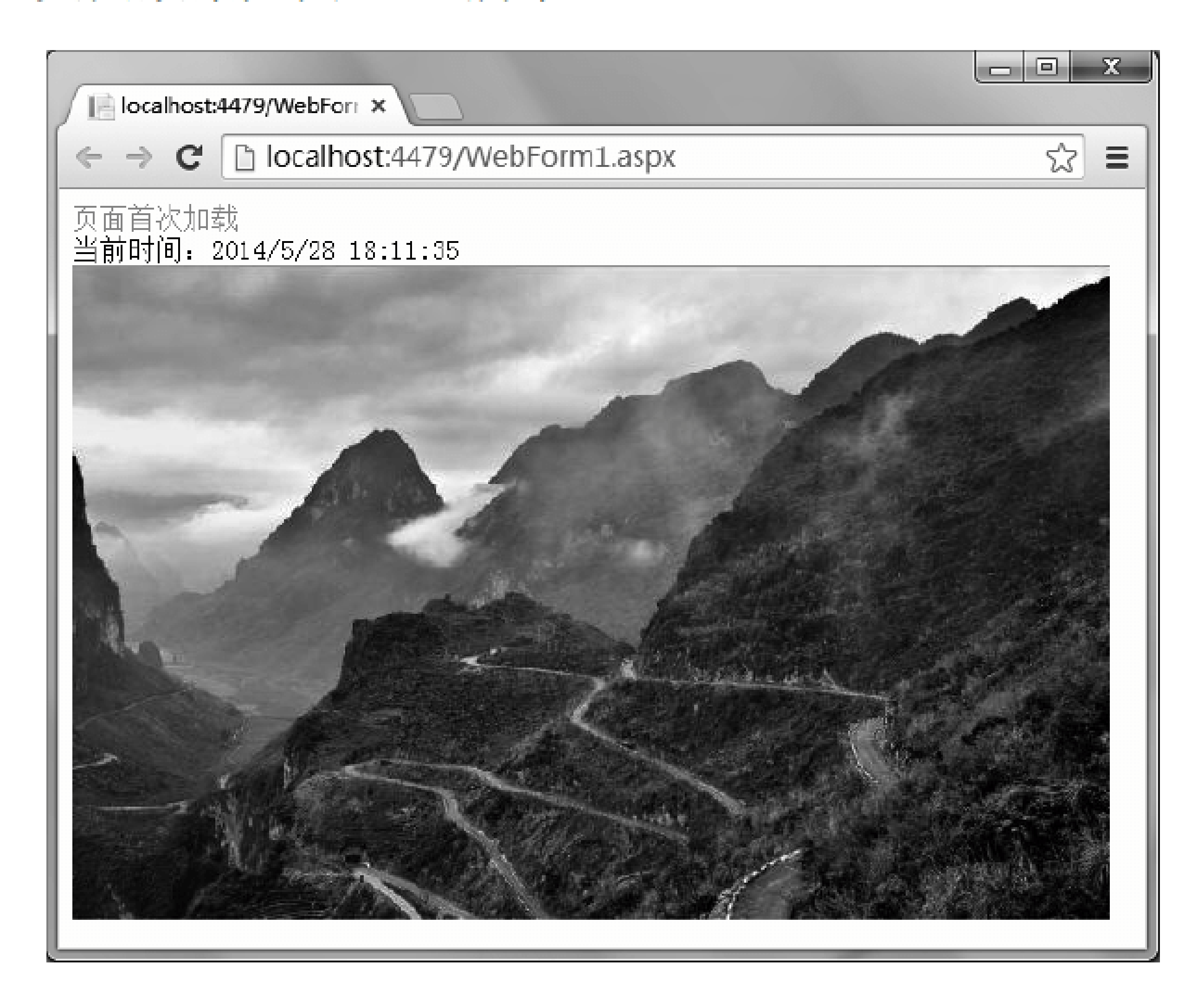


图 13–1　18:11:35 时刻的页面效果

（5）在页面运行中可以看出，无论当前时间标签和图片怎样变化，页面始终显示为“页面首次加载”。如在 18:11:36 时刻，页面效果如图 13-2 所示。

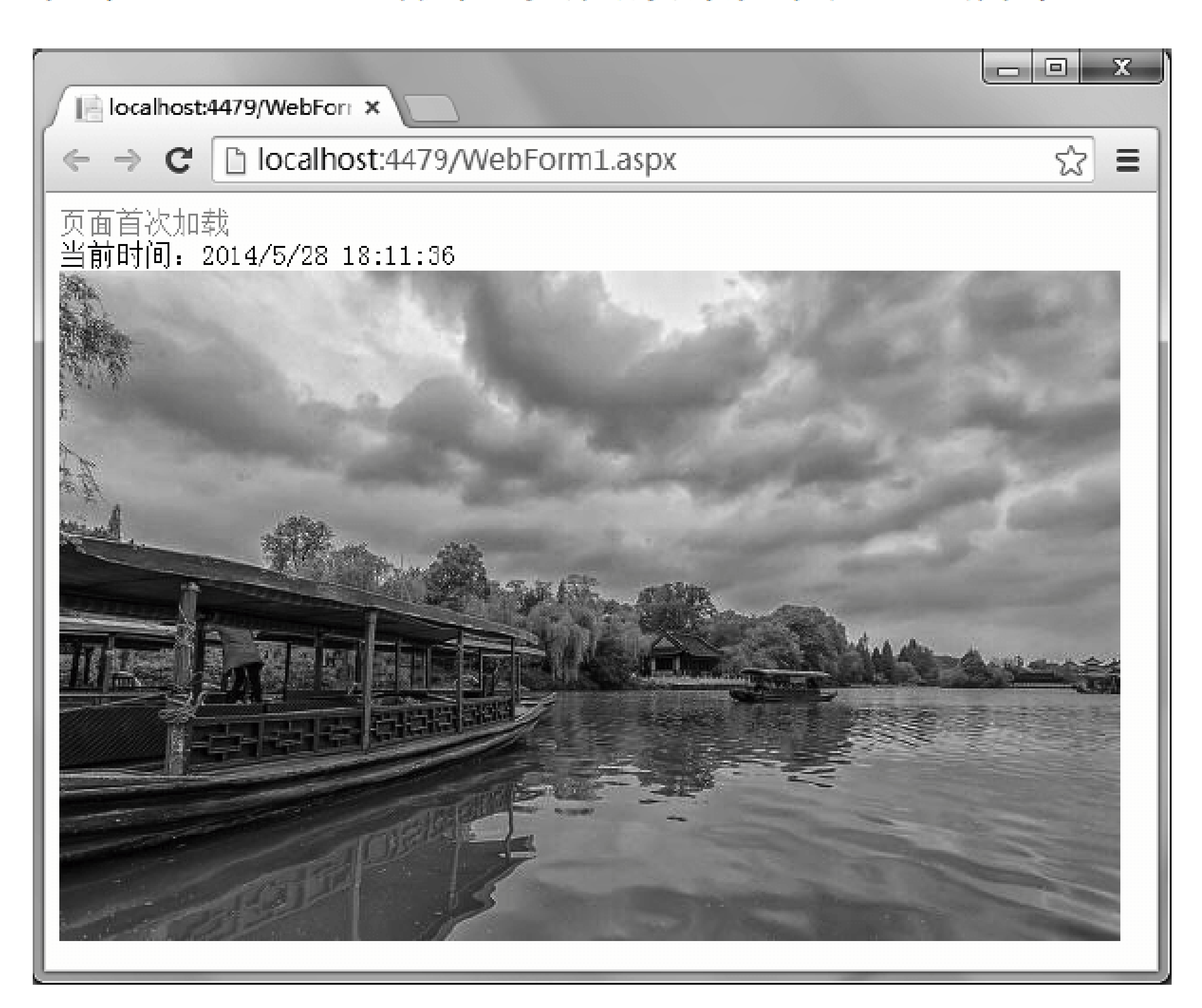


图 13–2　18:11:36 时刻的页面效果

13.6　jQuery

jQuery 是一个 JavaScript 库，使用 jQuery 极大地简化了 JavaScript 编程。JavaScript 是世界上最流行的脚本语言，而 jQuery 集成了 JavaScript 语法和函数，甚至可以在不了解 JavaScript 技术的情况下使用脚本。本节介绍 jQuery 的基础知识。

13.6.1 jQuery 简介

jQuery 很容易学习，jQuery 库包含以下特性：HTML 元素选取、HTML 元素操作、CSS 操作、HTML 事件函数、JavaScript 特效和动画、HTML DOM 遍历和修改、AJAX 和 Utilities。

jQuery 库位于一个 JavaScript 文件中，其中包含所有的 jQuery 函数。在使用 jQuery 之前需要在页面中引用该文件，放在页面的<head>标记下，如下所示。

```
<head>
<script src="jquery.js"></script>
</head>
```

上述代码中的 src 属性指出该页面所引用的 jQuery 库，该地址可以是网络中的地址，也可以是下载到本地的地址。通常可以在 http://jquery.com/download/网址下下载 jQuery 库，如图 13-3 所示。

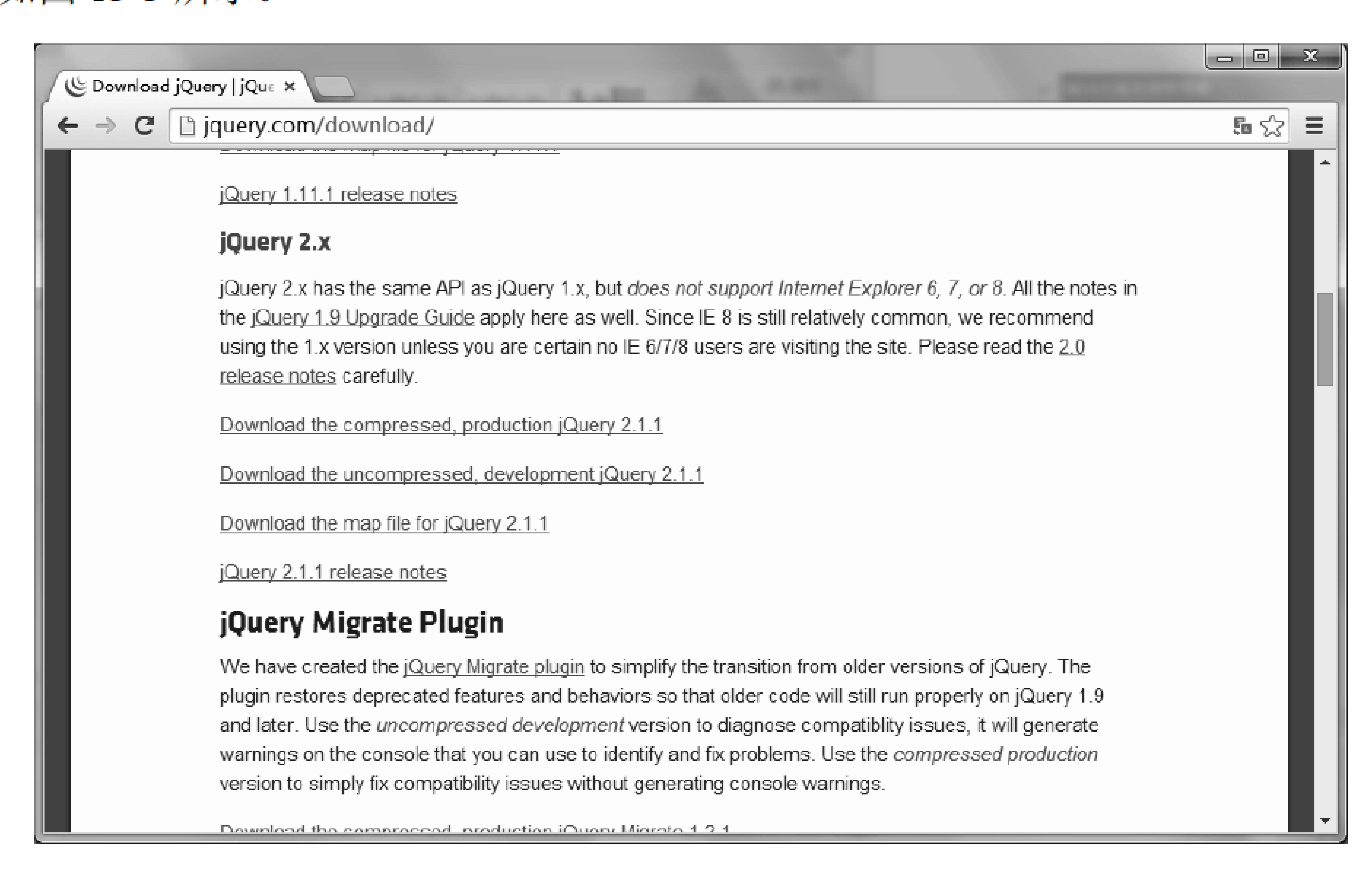


图 13-3　下载 jQuery

从图 13-3 可以看出，有两个版本的 jQuery 可供下载，如下所示。

（1）Production version：用于实际的网站中，已被精简和压缩。

（2）Development version：用于测试和开发（未压缩，是可读的代码）。

可以把下载文件放到与页面相同的目录中，方便使用。如果不希望下载并存放 jQuery，那么也可以通过 CDN（Contenr Delivery Network，内容分发网络）引用它。谷歌和微软的服务器都存有 jQuery 库，可以在官方网站上找到最新版本的 jQuery 来引用。如谷歌 1.8.0 版本的 jQuery，其地址为 http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js，使用如下代码引用。

```
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.
js">
</script>
```

jQuery 语法是为 HTML 元素的选取编制的，可以对元素执行某些操作。基础语法是：

```
$(selector).action()
```

对上述代码解释如下。

（1）美元符号定义 jQuery。

（2）选择符（selector）“查询”和“查找”HTML 元素。

（3）jQuery 的 action()执行对元素的操作。

例如，可使用下列语句执行指定操作。

（1）$(this).hide()：隐藏当前元素。

（2）$("p").hide()：隐藏所有段落。

（3）$(".test").hide()：隐藏所有 class="test"的元素。

（4）$("#test").hide()：隐藏所有 id="test"的元素。

jQuery 使用的语法是 XPath 与 CSS 选择器语法的组合。本节接下来将介绍更多有关选择器的语法。

13.6.2 jQuery 选择器

jQuery 学习的重点是如何准确地选取需要应用效果的元素，即 jQuery 选择器的定义。jQuery 元素选择器和属性选择器允许通过标签名、属性名或内容对 HTML 元素进行选择。以下分别从元素选择器、属性选择器和 CSS 选择器几个方面，来介绍选择器的使用。

1．jQuery 元素选择器

jQuery 使用 CSS 选择器来选择 HTML 元素，常见的有如下几种。

（1）$("p")选取<p>元素。

（2）$("p.intro")选取所有 class=“intro”的<p>元素。

（3）$("p#demo")选取所有 id=“demo”的<p>元素。

2．jQuery 属性选择器

jQuery 使用 XPath 表达式来选择带有给定属性的元素，常见的有如下几种。

（1）$("[href]")选取所有带有 href 属性的元素。

（2）$("[href='#']")选取所有带有 href 值等于“#”的元素。

（3）$("[href!='#']")选取所有带有 href 值不等于“#”的元素。

（4）$("[href$='.jpg']")选取所有 href 值以“.jpg”结尾的元素。

3．jQuery CSS 选择器

jQuery CSS 选择器可用于改变 HTML 元素的 CSS 属性。如把所有 p 元素的背景颜

色更改为红色，代码如下。

```
$("p").css("background-color","red");
```

4. 选择器实例

其他常见的选择器示例，如表 13-7 所示。

表 13-7 常见的选择器示例

语法	说明
$(this)	当前 HTML 元素
$("p")	所有<p>元素
$("p.intro")	所有 class="intro"的<p>元素
$(".intro")	所有 class="intro"的元素
$("#intro")	id="intro"的元素
$("ulli:first")	每个 的第一个 元素
$("[href$='.jpg']")	所有带有以".jpg"结尾的属性值的 href 属性
$("div#intro.head")	id="intro"的<div>元素中的所有 class="head"的元素

13.6.3 jQuery 事件

jQuery 是为事件处理特别设计的。事件处理程序指的是当 HTML 中发生某些事件时所调用的方法。jQuery 事件处理方法是 jQuery 中的核心函数，通常会把 jQuery 代码放到<head>部分的事件处理方法中。

如果一个网站包含许多页面，那么若希望 jQuery 函数易于维护，可以把 jQuery 函数放到独立的.js 文件中。表 13-8 列举了 jQuery 事件方法的一些例子。

表 13-8 常用的 jQuery 事件方法

Event 函数	绑定函数至
$(document).ready(function)	将函数绑定到文档的就绪事件（当文档完成加载时）
$(selector).click(function)	触发或将函数绑定到被选元素的单击事件
$(selector).dblclick(function)	触发或将函数绑定到被选元素的双击事件
$(selector).focus(function)	触发或将函数绑定到被选元素的获得焦点事件
$(selector).mouseover(function)	触发或将函数绑定到被选元素的鼠标悬停事件

jQuery 使用$符号作为 jQuery 的简介方式。某些其他 JavaScript 库中的函数（比如 Prototype）同样使用$符号，这样就可能产生冲突。jQuery 使用名为 noConflict()的方法来解决该问题。varjq=jQuery.noConflict()语句使用自己的名称（比如 jq）来代替$符号。

由于 jQuery 是为处理 HTML 事件而特别设计的，那么当遵循以下原则时，代码会更恰当且更易维护。

（1）把所有 jQuery 代码置于事件处理函数中。

（2）把所有事件处理函数置于文档就绪事件处理器中。

（3）把 jQuery 代码置于单独的.js 文件中。

（4）如果存在名称冲突，则重命名 jQuery 库。

13.6.4 jQuery 特效

jQuery 的常见应用是页面特效，在页面上实现动态的样式，如折叠菜单，图片的隐藏、显现，动画特效等。这些特效由 jQuery 效果方法来实现，如表 13-9 所示。

表 13-9 jQuery 效果方法

方法名称	说明
animate()	对被选元素应用“自定义”的动画
clearQueue()	对被选元素移除所有排队的函数（仍未运行的）
delay()	对被选元素的所有排队函数（仍未运行）设置延迟
dequeue()	运行被选元素的下一个排队函数
fadeIn()	逐渐改变被选元素的不透明度，从隐藏到可见
fadeOut()	逐渐改变被选元素的不透明度，从可见到隐藏
fadeTo()	把被选元素逐渐改变至给定的不透明度
hide()	隐藏被选的元素
queue()	显示被选元素的排队函数
show()	显示被选的元素
slideDown()	通过调整高度来滑动显示被选元素
slideToggle()	对被选元素进行滑动隐藏和滑动显示的切换
slideUp()	通过调整高度来滑动隐藏被选元素
stop()	停止在被选元素上运行动画
toggle()	对被选元素进行隐藏和显示的切换

以滑动方法为例，jQuery 拥有以下滑动方法：slideDown()、slideUp()和 slideToggle()，对其解释如下。

（1）slideDown()方法用于向下滑动元素，语法如下。

```
$(selector).slideDown(speed,callback);
```

对上述语法解释如下。

① 可选的 speed 参数规定效果的时长。它可以取以下值：slow、fast 或毫秒。

② 可选的 callback 参数是滑动完成后所执行的函数名称。

（2）slideUp()方法用于向上滑动元素，语法如下。

```
$(selector).slideUp(speed,callback);
```

① 可选的 speed 参数规定效果的时长。它可以取以下值：slow、fast 或毫秒。

② 可选的 callback 参数是滑动完成后所执行的函数名称。

（3）slideToggle()方法可以在 slideDown()与 slideUp()方法之间进行切换，语法如下。

```
$(selector).slideToggle(speed,callback);
```

① 如果元素向下滑动，则 slideToggle()可向上滑动它们。

② 如果元素向上滑动，则 slideToggle()可向下滑动它们。

思考与练习

一、填空题

1. Ajax 的核心是 JavaScript 的__________对象。

2. __________控件可以以指定间隔执行回发。

3. __________控件可以提供有关部分页呈现进度的反馈。

4. __________方法用于向下滑动元素。

5. $(selector).click(function)触发或将函数绑定到被选元素的__________事件。

6. jQuery 使用__________表达式来选择带有给定属性的元素。

二、选择题

1. 下列不属于 Visual Studio 2012 内置 Ajax 控件的是__________。

   A. ScriptManagerProxy
   B. UpdateProgress
   C. UpdatePanel
   D. AssociatedUpdatePanel

2. 下列描述错误的是__________。

   A. $("p")选取<p>元素
   B. $("p.intro")选取所有 class="intro"的<p>元素
   C. $("p.intro")选取所有 id="intro"的<p>元素
   D. $("p#demo")选取所有 id="demo"的<p>元素

3. ASP.NET 的 Ajax 功能只有在页面使用__________控件的基础上才能够实现。

   A. ScriptManager
   B. UpdateProgress
   C. UpdatePanel
   D. Timer

4. 下列不属于实现页面局部更新方法的是__________。

   A. RegisterPostBackControl()
   B. RegisterAsyncPostBackControl()
   C. RegisterGetBackControl()
   D. RegisterDispose()

5. 下列说法错误的是__________。

   A. slideDown()通过调整高度来滑动显示被选元素
   B. slideToggle()对被选元素进行滑动隐藏和滑动显示的切换
   C. slideUp()通过调整高度来滑动隐藏被选元素
   D. stop()停止在被选元素上进行滑动

6. slideDown()的 speed 参数规定效果的时长，其可取值不能是__________。

   A. slow
   B. normal
   C. fast
   D. 毫秒

7. Timer 控件 Interval 属性默认值是__________。

   A. 1s
   B. 10s
   C. 30s
   D. 60s

三、简答题

1. 简述异步回发的原理。
2. 总结不能够使用异步更新的情况。
3. 简单实现一个异步更新。
4. 总结 jQuery 选择器的使用。

第 14 章　Silverlight 入门

Silverlight 是一个跨浏览器的、跨平台的插件，为网络带来下一代基于.NET 的媒体体验和丰富的交互式应用程序。Silverlight 提供灵活的编程模型，并可以很方便地集成到现有的网络应用程序中。它可以对运行在 Mac 或 Windows 上的主流浏览器提供高质量视频信息的快速、低成本的传递。很多著名的 Web 应用都使用了 Silverlight，例如 QQ 体验版、PPlive 和 Sina 微博客户端等。

本章首先对 Silverlight 的概念进行介绍，然后重点讲解创建一个 Silverlight 应用程序的过程、Silverlight 的 XAML 以及与浏览器的交互。最后通过三个案例讲解 Silverlight 的简单应用。限于篇幅关系，本章不会对 Silverlight 的每个方面进行详细解释，有兴趣的读者可以查阅相关书籍。

本章学习要点：

- 了解什么是 Silverlight 及其与 WPF 的区别
- 掌握 Silverlight 应用程序的创建过程
- 熟悉 XAML 标记的语法规则、命名空间和后台文件
- 掌握 Silverlight 调用 HTML 的方法
- 掌握在 HTML 中调用 Silverlight 的方法
- 了解将 Silverlight 设置为桌面应用程序的方法

14.1　Silverlight 概述

Silverlight 中文翻译为银光，它是如今互联网 RIA 技术领域中的新宠。随着 Silverlight 版本的不断进化，其技术已经日趋成熟和健壮，并以其开放式的界面语言（XAML），优雅的编程语言（C#）等特点吸引了大批开发人员的目光。

14.1.1　Silverlight 简介

Microsoft Silverlight 是一种跨浏览器、跨平台的.NET Framework 实现，用于为 Web 生成和提供下一代媒体体验和丰富的交互式应用程序。Silverlight 统一了服务器、Web 和桌面的功能，统一了托管代码和动态语言、声明性编程和传统编程以及 WPF 的功能。

对于互联网用户来说，Silverlight 是一个安装简单的浏览器插件程序。用户只要安装了这个插件程序，就可以在 Windows 和 Mac 等多种浏览器中运行相应版本的 Silverlight 应用程序，享受视频分享、在线游戏、广告动画、交互丰富的网络服务等。

对于开发设计人员而言，Silverlight 是一种融合了微软的多种技术的 Web 呈现技术。它提供了一套开发框架，并通过使用基于向量的图像技术，支持任何尺寸图像的无缝整合，对基于 ASP.NET、Ajax 在内的 Web 开发环境实现了无缝连接。Silverlight 使开发设

计人员能够更好的协作，有效地创造出能在 Windows 和 Macintosh 上多种浏览器中运行的内容丰富、界面绚丽的 Web 应用程序——Silverlight 应用程序。

简而言之，Silverlight 是一个跨浏览器、跨平台的插件，为网络带来下一代基于.NET 媒体体验和丰富的交互式应用程序。对运行在 Macintosh 和 Windows 上的主流浏览器，Silverlight 提供了统一而丰富的用户体验，通过 Silverlight 这个小小的浏览器插件，视频、交互性内容，以及其他应用能完好地融合在一起。

Silverlight 将多种技术组合到单个开发平台，主要提供下列功能。

1．WPF 和 XAML

Silverlight 包含 WPF 技术的一个子集，从而大大扩展了浏览器中用于创建 UI 的元素。Silverlight 允许创建图形、动画、媒体和其他丰富的客户端功能，使基于浏览器的 UI 远超单独使用 HTML 提供的效果。XAML 提供用于创建元素的声明性标记语法。

2．对 JavaScript 的扩展

Silverlight 提供对通用浏览器脚本语言的扩展，可以控制浏览器 UI，包括使用 WPF 元素。

3．跨浏览器、跨平台支持

Silverlight 可以在所有通用浏览器（以及任意平台）上自由运行。开发人员可以设计和开发应用程序而不必担心用户具有何种浏览器或平台。

4．与现有应用程序集成

Silverlight 可以与现有 JavaScript 和 ASP.NET Ajax 代码无缝集成，以增强已具有的功能。

5．可以访问.NET Framework 编程模型

可以使用诸如 C#和 Visual Basic 的语言创建 Silverlight 应用程序。

6．工具支持

可以使用诸如 Visual Studio 和 Expression Blend 之类的开发工具快速创建 Silverlight 应用程序。

7．网络支持

Silverlight 包括对 TCP 上的 HTTP 的支持。可以连接到 WCF、SOAP 或 ASP.NET Ajax 服务并接收 XML、JSON 或 RSS 数据。

8．LINQ

Silverlight 包括语言集成查询（LINQ），这种查询允许使用直观本机语法和.NET Framework 语言中的强类型对象来编程进行数据访问。

14.1.2 Silverlight 结构

Silverlight 平台作为一个整体，主要由如下三部分组成。

1. 核心表示层框架

面向 UI 和用户交互的组件和服务（包括用户输入、用于 Web 应用程序的轻量型 UI 控件、媒体播放、数字版权管理和数据绑定）以及表示层功能（包括矢量图形、文本、动画和图像）。此外还包括用于指定布局的可扩展应用程序标记语言（XAML）。

2. .NET Framework for Silverlight

.NET Framework 中包含组件和库的一个子集，其中包括数据集成、可扩展 Windows 控件、网络、基类库、垃圾回收和公共语言运行时。

.NET Framework for Silverlight 的某些部分是通过应用程序部署的。这些 Silverlight 库是未包括在 Silverlight 运行时中但将在 Silverlight SDK 中提供的程序集。在应用程序中使用 Silverlight 库时，它们会与应用程序打包在一起，并下载到浏览器中。这些库包括新的 UI 控件、XLINQ、整合 RSS、XML 序列化和动态语言运行时。

3. 安装程序和更新程序

这是一个安装和更新控件，可简化用户首次安装该应用程序的过程，以及提供以后的自动更新功能。

如图 14-1 所示演示了 Silverlight 结构中这些组件及相关组件和服务。

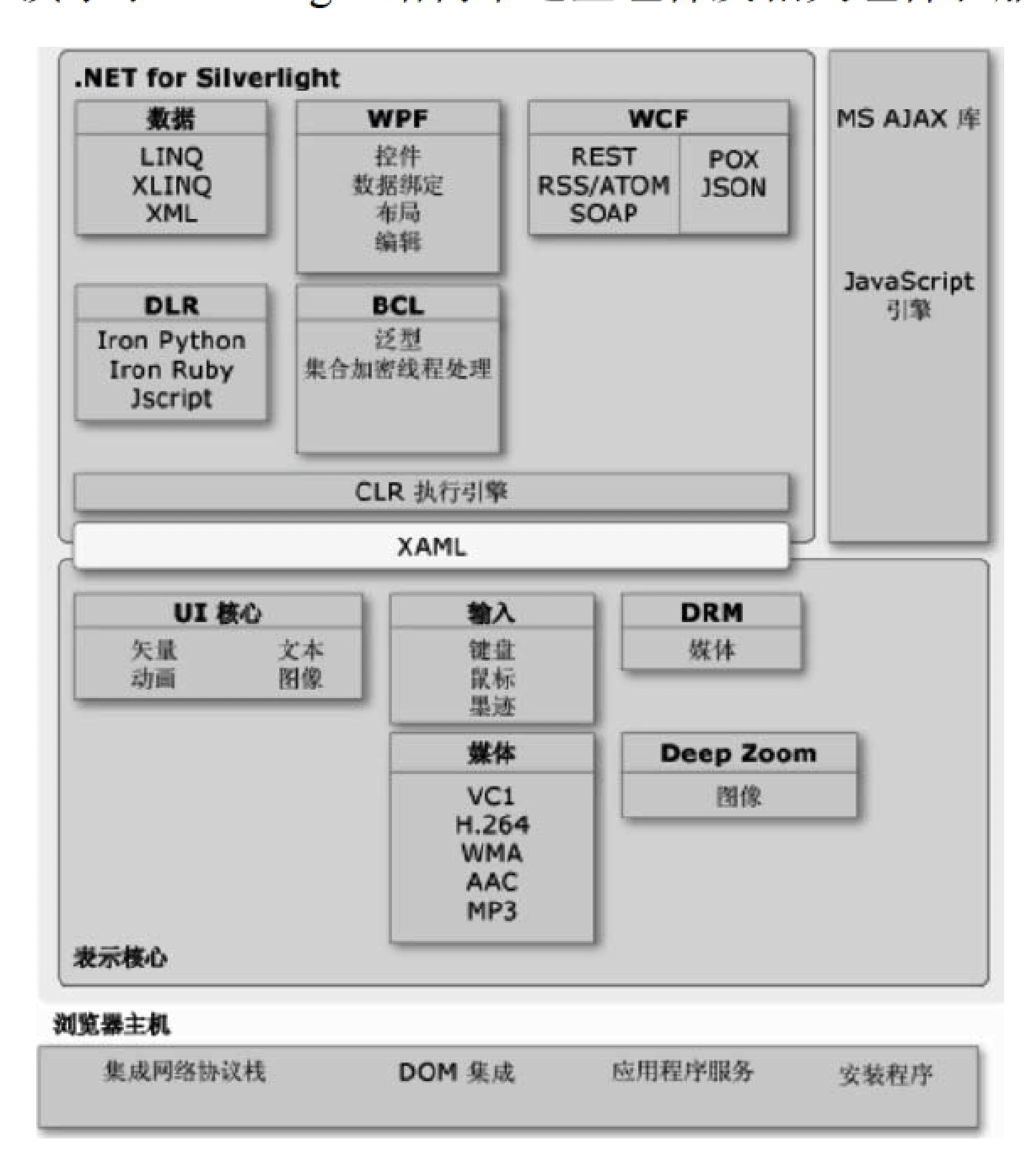


图 14-1 Silverlight 结构图

14.1.3　与 WPF 的比较

Silverlight 是 WPF 的一个子集，它提供了运行在 Web 浏览器中的应用程序。从 3.0 版本开始，Silverlight 应用程序也可以独立运行。

Silverlight 和 WPF 在许多方面都是类似的，但是两者之间也有重要的区别。Silverlight 由核心显示架构、.NET Framework for Silverlight、安装程序和更新程序组成；WPF 应用程序运行在 Windows 系统上，至少需要.NET Client Profile。Silverlight 使用一个插件模型，并驻留在 Web 浏览器中。除了 IE 之外，Silverlight 也可以用于 Firefox、Safari、Opera 和 Chrome 等浏览器中。

编译 WPF 应用程序时会得到一个可执行的程序集，其中包含二进制格式的 XAML 代码和资源。而对于 Silverlight 应用程序，编译器会创建一个 XAP 文件，这是一个 ZIP 包，其中包含程序集和配置信息。

.NET Framework for Silverlight 中的类与完整的.NET Framework 相同，但并不包含.NET Framework 中的所有类。前者删除了后者中的一些类，同时增加了一些适用于 Silverlight 的类。

从 Silverlight 3 开始不可以使用 WPF 中的流文档和固定文档。在 Silverlight 中可以用 2D 模拟 3D，但这完全不同于 WPF 的 3D 功能。与 Windows 窗体的交互操作功能也不适用。无论如何，最好用 XAML 编写新的 UI 界面，而不是集成 Windows 窗体控件。

WPF 与 Silverlight 应用程序的区别还体现在如下几点。

（1）画笔：Silverlight 没有 DrawingBrush、VisualBrush 和 TileBrush 画笔，但是可以使用 VideoBrush 画笔添加视频。

（2）字体：并不是所有字体都可以在所有平台上使用，所以 Silverlight 只能使用它内置的字体。

（3）控件：Silverlight 没有 Menu、ToolBar、Window 和 WebBrowser 控件。但是 Silverlight 提供的一些控件也不适用于 WPF。随着时间的推迟，这些区别会消失。

（4）联网：为了避免 UI 线程的阻塞，只有异步调用可用。在 Silverlight 4 中只能通过 WCF 使用 BasicHttpBinding 和 PollingDuplexHttpBinding，但还可以使用二进制编码。

（5）文件系统访问：Silverlight 3 应用程序不允许读写客户端系统，但可以读写独立的存储器。在运行 Silverlight 的控件上单击鼠标右键会出现配置菜单，其中的 Application Storage 选项卡提供了通过独立存储器使用数据的应用程序的相关信息，还可以删除它。

（6）浏览器集成：因为 Silverlight 控件一般运行于 Web 浏览器中，所以与浏览器的集成很重要。可以定义能从 JavaScript 中调用的.NET 类，使用 System.Windows.Browser 命名空间中的类还可以在 Silverlight 中调用 HTML 和 JavaScript。

（7）媒体：这是 Web 应用程序中得到 Silverlight 特别支持的一个重要方面。通过流，可以使用渐进下载和平滑流等功能，还可以给流添加时间轴标记。利用 DeepZoom 技术和 MultiScaleImage 控件，可以创建令人印象深刻的用户体验，允许用户放大从几个图像文件中构建的大型图像。

如表 14-1 所示列出了 WPF 与 Silverlight 在功能上的对比。

表 14-1 WPF 与 Silverlight 功能对比

功能名称	WPF	Silverlight
XAML	完整	完整
控件	完整	完整
布局	完整	完整
Binding	完整	基本完整
依赖属性	完整	基本完整
路由事件	完整	简化
命令	完整	无
资源	完整	完整
数据模板	完整	基本完整
绘图	完整	基本完整
2D/3D 动画	完整	简化

14.2 实验指导——创建第一个 Silverlight 应用程序

了解 Silverlight 的概念之后，现在使用 VS 2012 创建一个 Silverlight 程序。在 VS 2012 中支持 5 种类型的 Silverlight 项目，如图 14-2 所示。

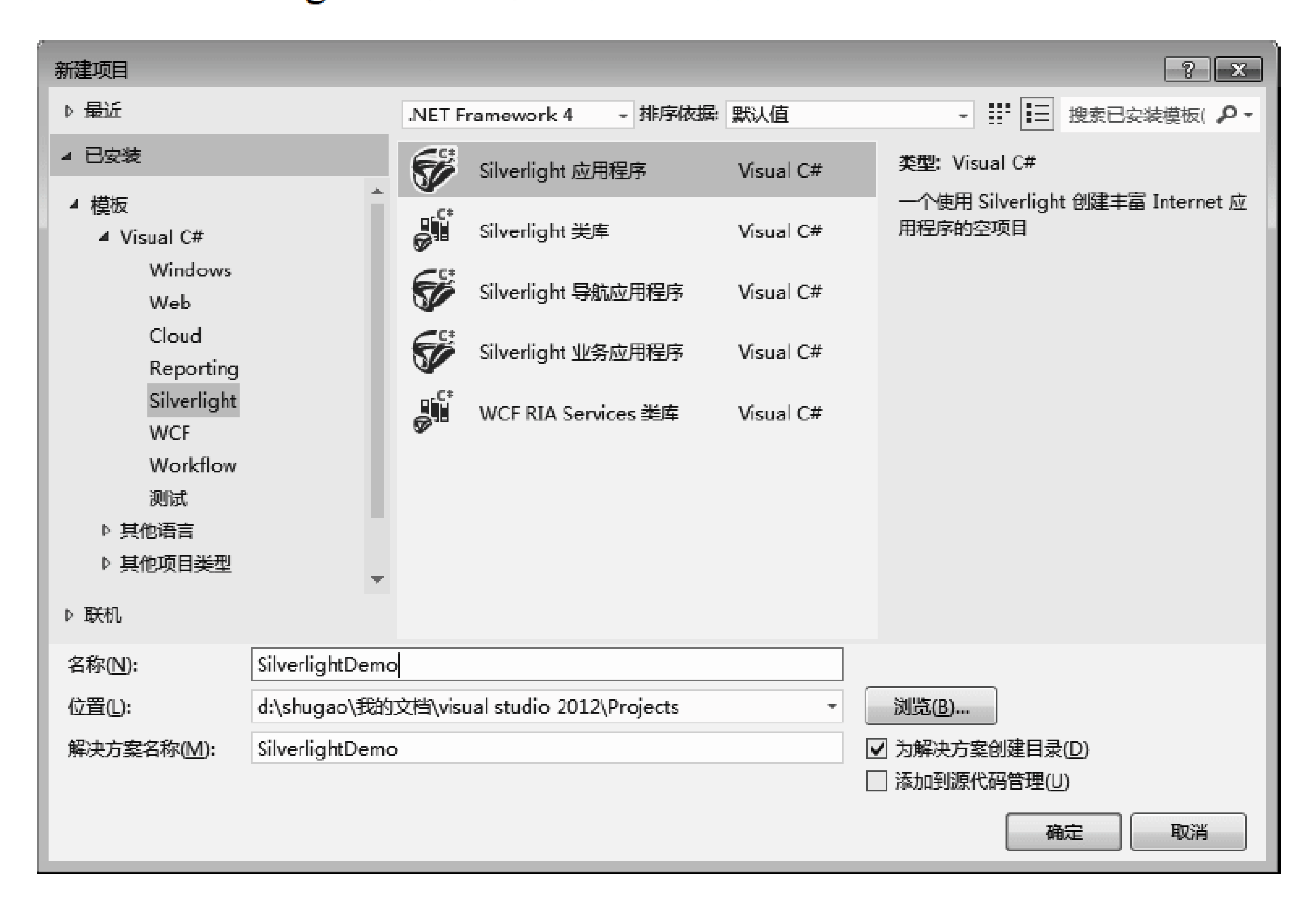


图 14-2 新建 Silverlight 项目

这里以 Silverlight 应用程序类型为例，具体步骤如下。

（1）在如图 14-2 所示的【新建项目】对话框中选择【Silverlight 应用程序】类型，将项目名称设置为 SilverlightDemo，再单击【确定】按钮。

（2）由于 Silverlight 应用程序必须在受主机托管的 HTML 页面中运行，所以需要选择一个新网站来承载 Silverlight 应用程序。新网站可以是 ASP.NET Web 应用程序项目或者 ASP.NET 网站。VS 2012 支持 Silverlight 4 和 Silverlight 5 两个版本，所以还需要选择

一个项目中使用的版本。

在这里使用 ASP.NET Web 应用程序项目作为承载项目类型，并选择版本为 Silverlight 4，再单击【确定】按钮，如图 14-3 所示。



图 14-3　新建 Silverlight 应用程序

（3）进入 Silverlight 应用程序的设计界面，该界面与 Windows 窗体程序设计界面类似，如图 14-4 所示。

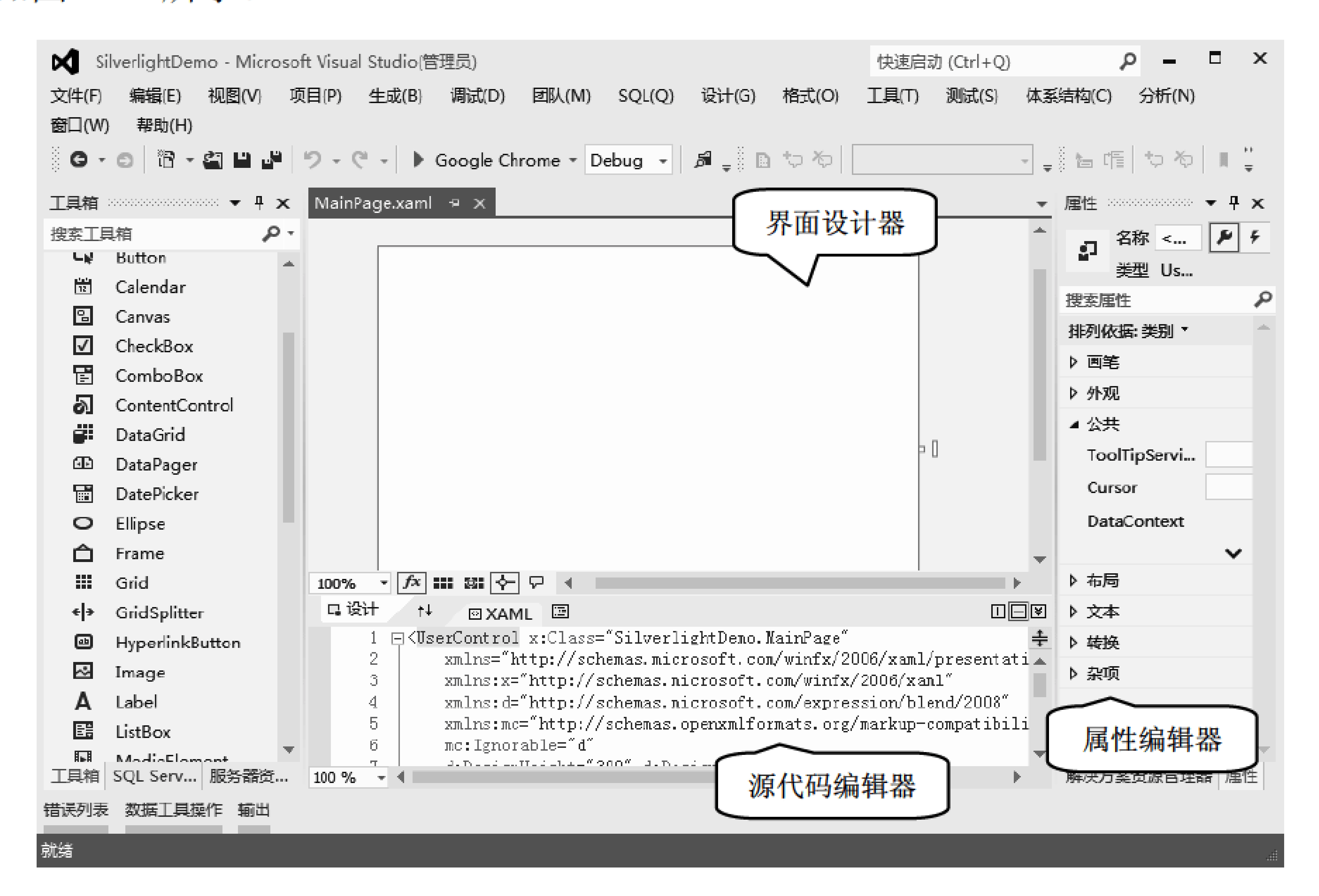


图 14-4　Silverlight 设计器

左边工具箱中罗列了所有可用的 WPF 控件，中间上方区域是程序的可视化界面设计器，下方是界面对应的 XAML 源代码编辑器，最右侧是属性编辑器。

从工具箱中拖动一个控件到界面设计器，同时在源代码编辑器中会自动添加相应的 XAML 代码。当然，对于熟练的开发人员也可以直接编写 XAML 代码，VS 2012 为代码

提供了非常完美的智能提示。为了查看设计时与 XAML 代码生成时的变化，可以拖动调整控件的位置和大小，以查看代码的变化。

（4）如图 14-5 所示为【解决方案资源管理器】窗格中当前 Silverlight 项目的文件结构。

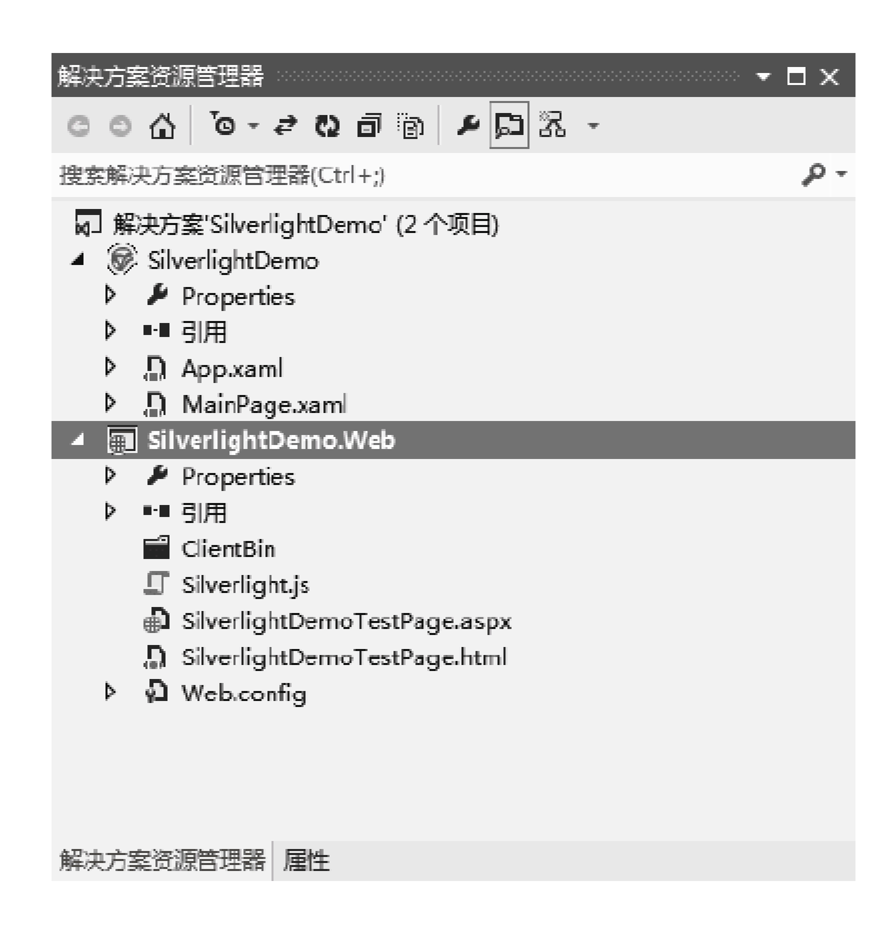


图 14–5　Silverlight 项目的文件结构

此时解决方案中包含两个项目，下面对它们进行简单介绍。

① SilverlightDemo 项目

这是一个 Silverlight 类型的项目，自动生成后会包含 App.xaml 和 MainPage.xaml 两个文件，这两个文件对应的后台文件是 App.xaml.cs 和 MainPage.xaml.cs。

其中，App.xaml 主要用来控制整个 Silverlight 应用程序的加载入口和异常的处理，并可以包含 Silverlight 应用程序的公共资源。MainPage.xaml 是 Silverlight 应用程序的默认用户控件，编译运行时会自动加载该用户控件到托管的 Web 页面中运行，它类似于 ASP.NET 中的 Default.aspx 页面。

② Silverlight.Web 项目

当创建 Silverlight 项目之后，在解决方案中会生成针对该项目的 Web 运行项目。在 Web 项目中包含 4 个文件，其中 Web.config 是网站的配置文件；Silverlight.js 是支持 HTML 页面中运行 Silverlight 的脚本文件。SilverlightDemoTestPage.aspx 和 SilverlightDemoTestPage.html 是两个测试 Silverlight 在 ASPX 和 HTML 页面中是否正常工作的文件。

（5）从工具箱中添加三个 Label 控件、一个 Button 控件和一个 TextBox 控件到 MainPage.xaml 的界面设计器，并调整其大小和位置。最终生成的 XAML 代码如下。

```
<UserControl  xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/
presentation/sdk"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    x:Class="SilverlightDemo.MainPage"
```

```
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="500">
    <Canvas >
        <sdk:Label Content="欢迎来到Silverlight的世界" FontSize="24" Canvas.
        Left="66" Canvas.Top="30" />
        <Button Content="确定" HorizontalAlignment="Left"  Vertical
        Alignment="Top" Width="75" Margin="66,158,0,0" Click="Button
        Click 1"/>
        <TextBox Name="txtName" HorizontalAlignment="Left" Height="23"
        TextWrapping="Wrap" VerticalAlignment="Top" Width="258" Margin=
        "66,121,0,0"/>
        <sdk:Label HorizontalAlignment="Left" Height="28" Margin="66, 95,
        0,0" VerticalAlignment="Top" Width="120" Content="请输入姓名:  "/>
        <sdk:Label Name="lblResult" HorizontalAlignment="Left" Height=
        "28" Margin="66,206,0,0" VerticalAlignment="Top"  Content=""/>
    </Canvas>
</UserControl>
```


（6）进入后台文件 MainPage.xaml.cs，为【确定】按钮添加单击事件处理方法，代码如下。

```
private void Button Click 1(object sender, RoutedEventArgs e)
 {
     lblResult.Content = "您好，" + txtName.Text + "。这是创建的第一个 Silver
     light 应用程序。";
 }
```

（7）右击解决方案选择【生成解决方案】命令，然后运行 SilverlightDemo.Web 项目中的两个测试文件查看运行效果，如图 14-6 所示。



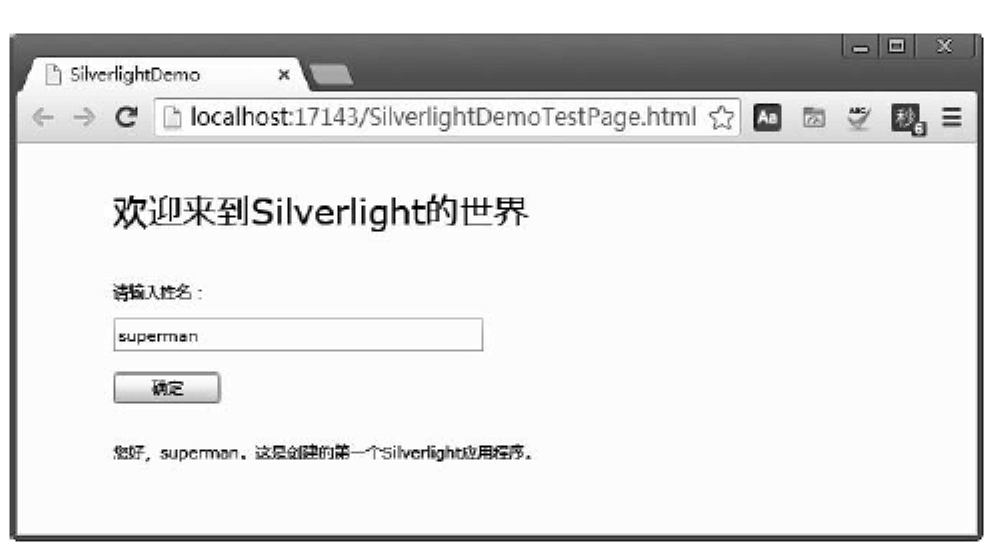


图 14-6　测试 Silverlight 程序

14.3　了解 XAML

XAML（eXtensible Application Markup Language）是声明式语言，用于将构成 XML 的各个元素组成一个应用程序，而这些元素可用来表示一个结构化的.NET 类集合。下面对 XAML 的概念进行详细介绍。

14.3.1 XAML 简介

使用传统的显示技术时，要想从代码中分离出图形内容并不容易。对于 Windows 窗体应用程序关键问题是创建的每个窗体都是由 C#代码定义的。当把控件拖动到设计器并配置控件时，VS 会在相应的窗体类中自动调整代码。但是图形设计人员没有任何可以使用 C#代码的工具。

相反，对于美工人员他们必须将工作内容导出为位图。然后可以使用这些位图确定窗体、按钮以及控件的外观。对于简单的固定用户界面这种方法还可行，但是在一些复杂布局界面中，将会变得非常麻烦和受限制。

而 Silverlight 通过 XAML 解决了这一问题。当在 VS 2012 中设计一个 Silverlight 应用程序时，窗口不会被转换为代码，而是转换为一系列的 XAML 标记。当运行应用程序时，这些标记用于生成构成用户界面的对象。

例如，要创建一个按钮则可以使用下面的 XAML。

```
<Button HorizontalAlignment="Left" Margin="95,85,0,0" VerticalAli gn
ment="Top" Width="75" Click="Button_Click_1">我要登录</Button>
```


XAML 中的元素名为 CLR 中的类名，上面的 Button 元素对应的是 Silverlight 中的 Button 类。XAML 的属性是相应类中的属性，例如上面 HorizontalAlignment、Margin 和 Width 等实际上是 Button 类中对应的属性。上述代码中，还定义了事件处理程序“Click="Button_Click_1"”，表示触发按钮 Click 事件时执行 Button_Click_1()方法，该方法在后台文件中进行了定义。

14.3.2 XAML 语法规则

首先介绍一下 XAML 的基本语法规则，如下所示。

（1）在 XAML 中的所有元素都映射为一个.NET 类的实例。元素的名称也完全对应于类名。例如，UserControl 元素表示在 WPF 中创建一个 System.Windows.Windows.UserControl 对象。

（2）与所有 XML 文档一样，可以在一个元素中嵌套另一个元素。例如，在 Grid 元素中嵌套一个 Button 元素。

（3）可以通过属性设置每个类的特征。但是，在某些情况下属性不能实现时，则需要通过特殊的语法使用嵌套的标签。

接下来看一个 XAML 文档的基本框架，如下所示。为了便于说明，对每行代码都添加了行号。

```
<UserControl x:Class="SilverlightDemo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
5      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6      mc:Ignorable="d"
7      d:DesignHeight="300" d:DesignWidth="400">
8      <Grid x:Name="LayoutRoot" Background="White">
9      </Grid>
10  </UserControl>
```

在这个文档中只包含两个元素，一个根级的 UserControl 元素和一个 Gird 元素。其中 UserControl 元素表示 Silverlight 应用程序的窗口，Grid 元素表示一个容器，在该容器内可以放置其他控件。

与在所有 XML 文档中一样，在 XAML 文档中有且只能有一个根级元素。在上面示例中根元素为 UserControl，在第 10 行使用</UserControl>标记结束了 UserControl 元素，此时文档也就结束了。也意味着在</UserControl>标记之后不允许有任何其他内容。

在第 1 行的 UserControl 元素开始标记中 Class 指定 Silverlight 程序对应的后台类名，第 2 行至第 5 行是 4 个 XML 命名空间。第 7 行定义了两个属性值，如下所示。

```
7      d:DesignHeight="300" d:DesignWidth="400">
```

表示的含义为当前 Silverlight 窗口的宽度为 400 像素，高度为 300 像素。

14.3.3 XAML 命名空间

我们知道，XAML 是从 XML 派生而来的。在 XML 中可以使用 xmlns 属性来定义命名空间，xmlns 的全称是 XML NameSpace。定义命名空间的好处是当来源不同的类重名时，可以使用命名空间来区分。

xmlns 属性的语法格式如下。

```
xmlns[:可选映射前缀]="命名空间 URL"
```

xmlns 关键字后跟一个可选的映射前缀，之间用冒号分隔。如果没有设置映射前缀，那默认所有来自该命名空间的标记前都不用加前缀名，没有前缀名的命名空间也称为默认命名空间。一个 XAML 文档只能有一个默认命名空间，通常选择元素最频繁使用的命名空间作为默认。

例如，在 14.3.2 节的 XAML 文档代码中 UserControl 元素和 Grid 元素都来自第二行声明的默认命名空间。而第 6 行中的 DesignHeight 属性则来自 d 前缀引用的命名空间，该命名空间在第 4 行进行了定义。

【范例 1】

对 14.3.2 节的 XAML 文档进行修改，将默认命名空间修改为 m 前缀，其他命名空间不变。修改后的代码如下。

```
<m:UserControl x:Class="SilverlightDemo.MainPage"
    xmlns:m="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
    <m:Grid x:Name="LayoutRoot" Background="White">
    </m:Grid>
</m:UserControl>
```

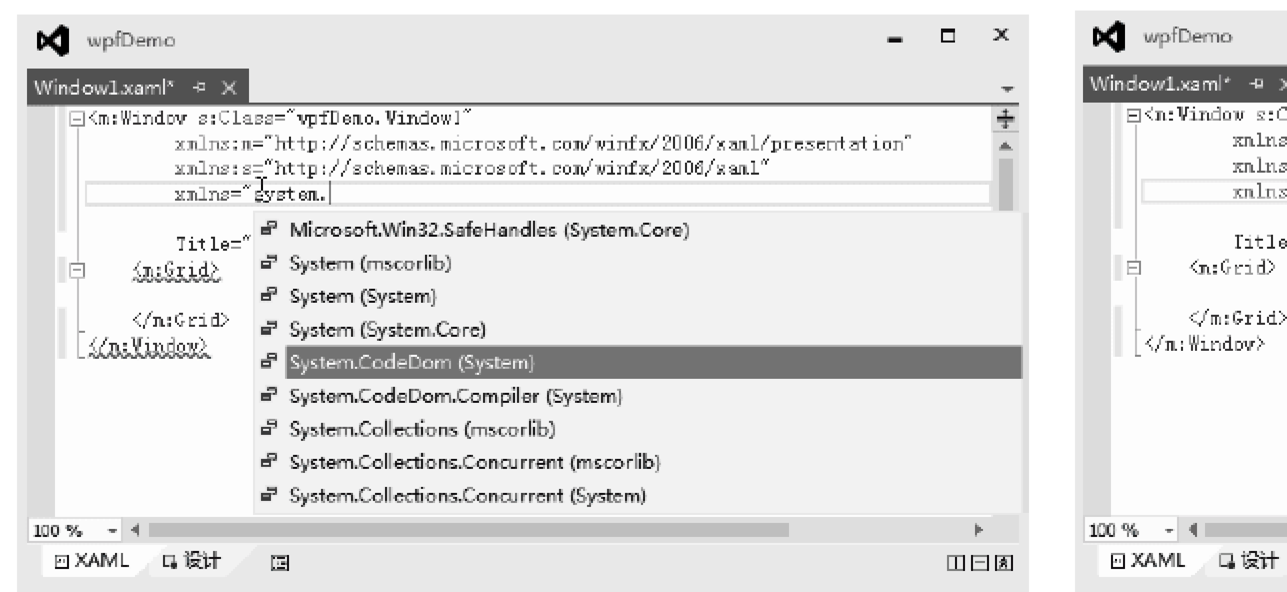
在上述代码中，第二行引用命名空间时指定了前缀为 m，因此该命名空间下的 UserControl 元素和 Grid 元素应该使用 m:UserControl 和 m:Grid 进行标记。

在 XAML 中引用其他程序集和.NET 命名空间的语法与 C#是不一样的。在 C#中，如果想使用 System.Windows.Controls 命名空间里的 Button 类，需要先把该命名空间所在的程序集通过添加引用的方式引用到项目中，然后在 C#代码的顶部使用“using System.Windows.Controls;”语句引入该命名空间。在 XAML 中实现相同的功能也需要添加对程序集的引用，然后再到根元素的开始标记中通过如下语句进行引用。

```
xmlns:c="clr-namespace:System.Windows.Controls;assembly=PresentationFramework"
```

其中，c 是命名空间的前缀，也可以是其他任意符合规范的名称。

命名空间 URL 并不是一个真实存在的 URL 地址，在这里只是 XAML 解析器的一个硬性编码，即看到这个 URL 就会引入一系列的程序集和程序集中包含的.NET 命名空间。无论是程序集还是命名空间 URL，用户都不需要去死记，因为 VS 2012 提供了强大的智能提示，如图 14-7 所示为输入“xmlns”之后的智能提示。



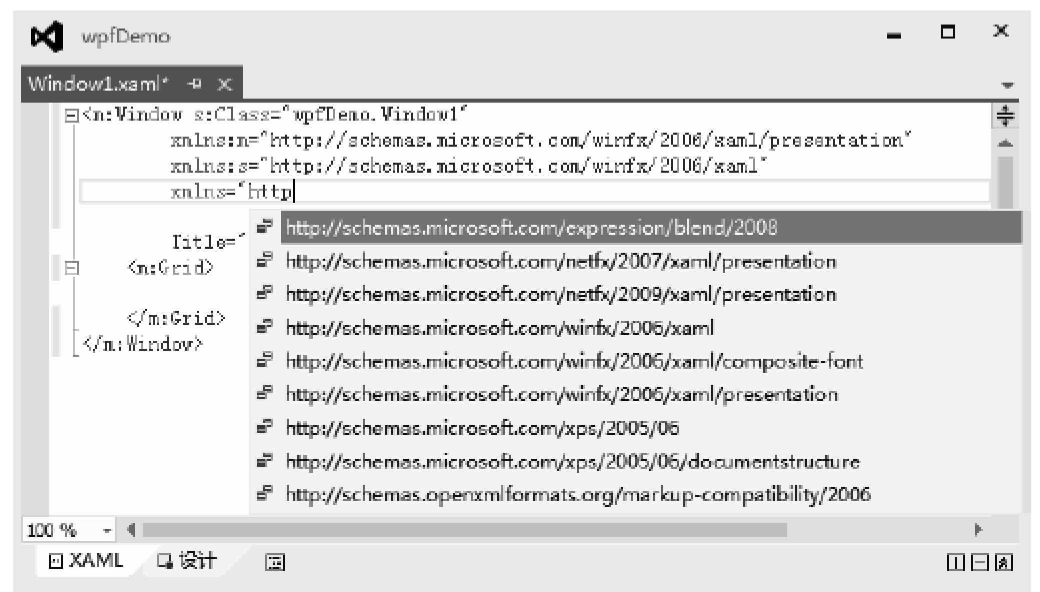


图 14–7 VS 2012 的程序集和命名空间智能提示

14.3.4 XAML 后台文件

与 ASP.NET 一样，XAML 也使用代码分离技术。一个 Silverlight 应用程序通常由两大部分组成，一部分是使用 XAML 描述 UI 元素在界面上的位置、大小等属性；另一部分是用来处理程序的逻辑、对传递事件的响应等操作。

在 XAML 中通过根元素的 Class 属性指定后台文件。例如，在 MainPage.xaml 文件中有代码“x:Class="SilverlightDemo.MainPage"”，此时 VS 2012 会自动创建一个后台文件 MainPage.xaml.cs，并生成如下部分类代码。

```
namespace SilverlightDemo
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }
    }
}
```

上述代码创建一个名为 MainPage 的类，该类继承是 XAML 中的 UserControl 基类。该类创建时使用 partial 关键字指定这是一个部分类。在该类中仅包含一个构造函数，构造函数又调用了 InitializeComponent()方法。

InitializeComponent()方法在 Silverlight 应用程序中扮演着非常重要的角色。InitializeComponent()方法在源代码中是不可见的，因为它是在编译应用程序时自动生成的。本质上，InitializeComponent()方法的所有工作是调用 System.Windows.Application 类的 LoadComponent()方法。LoadComponent()方法从程序集中提取编译过的 XAML，并使它来构造用户界面。当解析 XAML 时，它会创建每个控件对象，设置其属性，并关联所有事件处理程序。

14.4 与浏览器交互

在最初的 Silverlight 1 中只能使用 JavaScript 来编程，从 Silverlight 2 开始改变了这一状况。但是仍然有很多情况下需要在 Silverlight 中控制 HTML 和 JavaScript 代码，或者从 JavaScript 中调用.NET 方法。这两种情况可以使用 System.Windows.Browser 命名空间来处理，下面介绍具体的实现方法。

14.4.1 调用 HTML 页面

为了测试从 Silverlight 中调用 HTML 页面，首先需要创建一个 Silverlight 项目，再到 XAML 中布局和编写调用 HTML 代码，最后在 HTML 页面中设置显示位置。具体步骤如下。

【范例 2】

（1）新建一个名为 SilverlightApplication1 的 Silverlight 应用程序。

（2）打开 MainPage.xaml 在界面设计器上使用 Grid 控件制作一个 4 行 2 列的表格，并向单元格中填充内容。最终代码如下所示。

```
<Grid Background="#FF28ABB2">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="auto"></ColumnDefinition>
        <ColumnDefinition Width="*"></ColumnDefinition>
```

```
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"></RowDefinition>
        <RowDefinition Height="Auto"></RowDefinition>
        <RowDefinition Height="Auto"></RowDefinition>
        <RowDefinition Height="Auto"></RowDefinition>
    </Grid.RowDefinitions>
    <TextBlock Margin="3" Grid.Column="1" FontSize="20">用户注册</Text
     Block>
    <TextBlock Margin="3" Grid.Row="1">用户昵称：</TextBlock>
    <TextBox Height="23" Width="200" HorizontalAlignment="Left" Grid.
     Column="1" Grid.Row="1" Name="txtName" />
    <TextBlock Margin="3" Grid.Row="2">用户密码：</TextBlock>
    <TextBox Height="23"  Width="200" HorizontalAlignment="Left" Grid.
     Column="1" Grid.Row="2" Name="txtPass"/>
    <Button  Grid.Column="1" Grid.Row="3" Content="确定" Horizontal
     Alignment="Left" Width="75" Click="button1_Click"/>
</Grid>
```

上述代码主要包含两个用于输入的 TextBox 控件以及一个用于提交的 Button 控件。

（3）在 MainPage.xaml.cs 文件中编写按钮的单击事件处理方法 button1_Click，代码如下所示。

```
private void button1_Click(object sender, RoutedEventArgs e)
 {
    HtmlDocument doc = HtmlPage.Document;  //创建一个表示 HTML 页面的对象
    HtmlElement el1 = doc.GetElementById("dvResult");
                                      //获取 HTML 页面上的对象
    string html = "<ul><li>用户名：" + txtName.Text + "</li><li>用户密码：" +
    txtPass.Text + "</li></ul>";   //获取用户输入的内容，并组成 HTML 字符串
    el1.SetProperty("innerHTML", html);    //插入内容
 }
```

上述代码是 Silverlight 调用 HTML 页面的核心代码，首先通过 HtmlPage.Document 属性创建一个表示 HTML 页面的对象 doc，再调用该对象的 GetElementById()方法从 HTML 页面获取 ID 为 dvResult 的对象，然后通过 SetProperty()方法设置显示的 HTML 代码。整个过程与使用 HTML 中的 DOM 模型非常类似。

注意

必须先使用“using System.Windows.Browser”语句引入 HtmlPage 所在的命名空间，否则将出错。

（4）经过上面的步骤 Silverlight 项目的工作就完成了。下面打开 Web 项目中用于测试 Silverlight 的 HTML 页面，并在<form>标记下添加如下代码。

```
如下是由 Silverlight 中返回的用户信息：
<div id="dvResult"></div>  <hr/>
```

上述代码中最主要的是 id 为 dvResult 的元素，因为最终由 Silverlight 返回的内容将显示到该元素内。

（5）编译 Silverlight 项目，并运行 HTML 测试页面。首次运行时，由于没有设置任何内容显示效果如图 14-8 所示。

（6）在 Silverlight 中输入用户名和密码，再单击【确定】按钮，此时显示效果如图 14-9 所示。



图 14-8 首次运行时

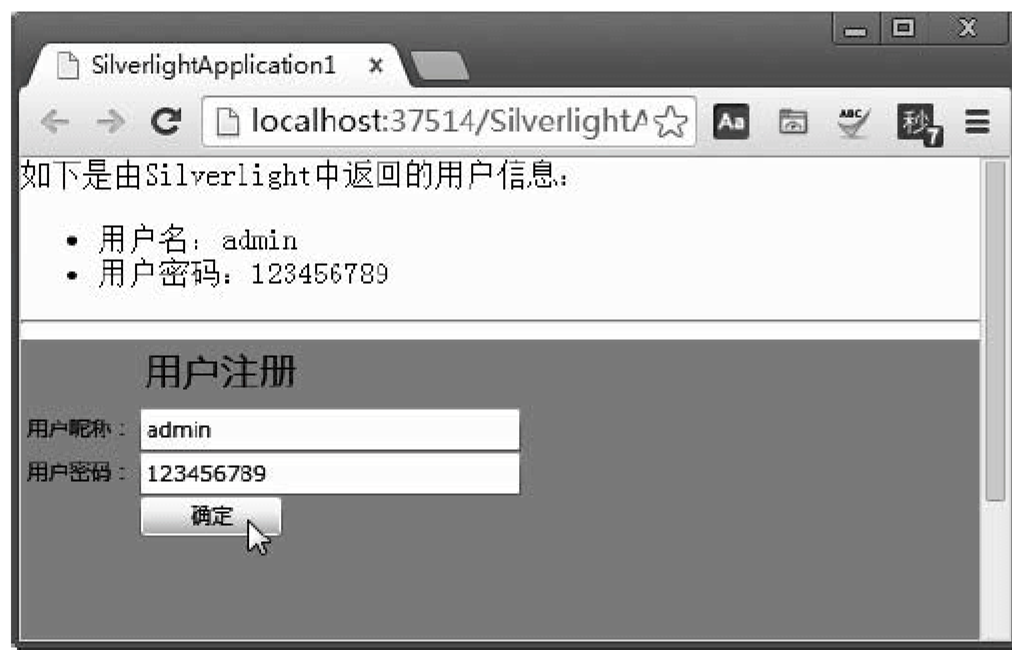


图 14-9 调用效果

提 示

使用 System.Windows.Browser 命名空间中的类可以访问 HTML 中的所有信息。其中最常用的 HtmlPage 类、HtmlDocument 类、HtmlWindow 类和 HtmlObject 类，分别用于访问浏览器信息、Cookie 页面中的元素和样式表，可以读取和修改内容。

14.4.2 调用 Silverlight

如果希望在 HTML 页面可以调用 Silverlight 有两种方法，第一种是使用 HtmlElement 对象的 AttachEvent()方法，另一种是使用 ScriptableMember 特性。下面分别介绍这两种方法的具体实现过程。

【范例 3】

假设在 Silverlight 项目的 MainPage.xaml 中有如下的布局代码。

```
<Canvas Background="#FFC96767">
    <TextBox Name="textbox1"  Height="23" TextWrapping="Wrap" Vertical
     Alignment="Top" Text="TextBox1" Width="258" Canvas.Left="59" Canvas.
     Top="36"/>
    <Button Content="同步"  Width="75" Click="Button_Click_1" Canvas.
    Left="59" Canvas.Top="80"/>
    <TextBox Name="textbox2" Height="23" Canvas.Left="59" Text Wrapping=
    "Wrap" Text="TextBox2"   Width="258" Canvas.Top="131"/>
</Canvas>
```

在后台文件已经实现了单击【同步】按钮时将 textbox1 中的内容复制到 textbox2 中。

现在要实现单击 HTML 页面中的一个按钮完成同步功能，步骤如下。

（1）在测试 Silverlight 的 HTML 页面中使用如下代码添加一个用于单击的按钮。

```
<input type="button" value="在 HTML 中实现同步功能" id="button1" />
```

上述代码创建一个 id 为 button1，值为“在 HTML 中实现同步功能”的 HTML 按钮。

（2）进入 MainPage.xaml 的后台 cs 文件，在构造函数中找到 HTML 按钮 button1，并注册单击事件处理函数。具体实现代码如下。

```
public MainPage()
{
    InitializeComponent();
    HtmlDocument doc = HtmlPage.Document;   //获取 HTML 页面
    HtmlElement button1 = doc.GetElementById("button1");
                                            //找到 button1
    button1.AttachEvent("onclick", onHtmlButtonClick);
                                            //为 onclick 事件注册处理函数
}
```

上述代码实现了单击 HTML 页面 button1 时执行 onHtmlButtonClick()函数。

（3）编写 onHtmlButtonClick()函数，实现复制功能。具体实现代码如下。

```
private void onHtmlButtonClick(object sender, HtmlEventArgs e)
{
    textbox2.Text = textbox1.Text;      //使 textbox2 的内容与 textbox1 相同
}
```

（4）现在运行 HTML 页面，单击【在 HTML 中实现同步功能】按钮查看 onHtmlButtonClick()函数执行效果。

【范例 4】

为了使用 ScriptableMember 特性，需要在 Silverlight 后台文件中将可以被 HTML 页面调用的方法添加“[ScriptableMember]”进行修饰。假设，以 MainPage.xaml.cs 中有一个 ToUpper()方法用于将字符串转换为大写，应用 ScriptableMember 特性后的代码如下所示。

```
[ScriptableMember]
public string ToUpper(string s)     //转换大写方法，参数 s 为要转换的字符串
{
    return s.ToUpper();             //返回转换后的大写字符串
}
```

在这里要注意，应用 ScriptableMember 特性的方法必须具有 public 作用域，且带有一个返回值。

如果一个类中包含可在 HTML 页面中调用的脚本，还必须在后台进行注册，以便 Silverlight 脚本引擎能找到该类型。注册方法是调用 HtmlPage 类的 RegisterScriptableObject()方法。该方法的第一个参数是 JavaScript 中用于查找对象的键

名，第二个参数定义了用于访问的成员实例（通常为 this）。

在本示例中使用的注册代码如下。

```
public MainPage()
{
    InitializeComponent();
    HtmlPage.RegisterScriptableObject("ScriptKey", this);
                                              //指定一个键名来注册脚本
}
```

现在便可以在 HTML 中访问 Silverlight 中的 ToUpper()方法了。为了使 HTML 能找到 ToUpper()方法，首先要找到 Silverlight 控件，这需要对默认生成的代码进行修改。为测试 Silverlight 的 object 标记添加一个 id 属性，并同时添加用于测试的 HTML 代码，最终代码如下所示。

```
请输入一个字符串：<input type="text" id="inWords" />
<input type="button" value="转换" onclick="transfer()" /><br />
转换结果如下：<div id="dvResult"></div>
<div id="silverlightControlHost">
    <object id="plugin" data="data:application/x-silverlight-2," type=
    "application/x-silverlight-2" width="100%" height="100%">
        <param name="source" value="ClientBin/SilverlightAppli cation2.
        xap" />
        <param name="onError" value="onSilverlightError" />
        <param name="background" value="white" />
        <param name="minRuntimeVersion" value="5.0.61118.0" />
        <param name="autoUpgrade" value="true" />
        <a href="http://go.microsoft.com/fwlink/?LinkID=149156&v= 5.0.
        61118.0" style="text-decoration: none">
            <img src="http://go.microsoft.com/fwlink/?LinkId=161376" alt=
            "获取 Microsoft Silverlight" style="border-style: none" />
        </a>
    </object>
    <iframe id="_sl_historyFrame" style="visibility: hidden; height: 0px;
    width: 0px; border: 0px"></iframe>
</div>
```

当单击【转换】按钮之后会调用 HTML 的 transfer()方法，该方法通过 id 属性找到 Silverlight，再使用指定的键名获取后台文件的引用，接着调用 ToUpper()方法并显示转换后的结果。最终代码如下所示。

```
function transfer() {
     var ClientObject = document.getElementById("plugin");
     var ServerObject = ClientObject.content.ScriptKey;
     var str = document.getElementById("inWords").value;
     var result = ServerObject.ToUpper(str);
     document.getElementById("dvResult").innerHTML = result;
 }
```

最后编译整个 Silverlight 项目，然后运行 HTML 页面测试转换效果，这里不再演示。

14.5 实验指导——创建脱离浏览器的桌面应用程序

从 Silverlight 3 版本开始，Silverlight 应用程序也可以在浏览器之外运行。此时的 Silverlight 应用程序可以由没有管理员权限的用户安装。但是 Silverlight 应用程序仍仅限于安全沙箱，在客户端系统上不能被完全信任。与 WPF 相比，在浏览器外运行的 Silverlight 应用程序的优势是不用安装.NET Framework，只需安装 Silverlight 运行库即可，而且通过 Silverlight 插件可以实现跨平台运行。

下面以 14.2 节创建的 Silverlight 应用程序为例，讲解如何使 Silverlight 应用程序脱离浏览器进行运行。

（1）在 VS 2012 中打开 SilverlightDemo 解决方案。

（2）在【解决方案资源管理器】窗格中右击 SilverlightDemo 项目，选择【属性】命令，进入项目属性设置面板。

（3）从面板中启用【允许在浏览器外运行应用程序】复选框，如图 14-10 所示。

（4）单击【浏览器外设置】按钮，在弹出的对话框中对应用程序的标题、大小和图标等选项进行设置，如图 14-11 所示。



图14-10 属性设置面板



图14-11 设置应用程序

（5）设置完成之后单击【确定】按钮再保存项目，并重新编译解决方案。

（6）运行 SilverlightDemo.Web 应用程序查看上述设置后 Silverlight 应用程序运行效果。在浏览器任意位置右击，都将会看到一个包含【将 SilverlightDemo 应用程序安装到此计算机】项的菜单，如图 14-12 所示。



图14-12　查看 Silverlight 应用程序运行效果

（7）选择【将 SilverlightDemo 应用程序安装到此计算机】项，弹出【安装应用程序】对话框。在该对话框中显示了应用程序的图片，同时还可以指定是否在【开始】菜单和桌面添加快捷方式，如图 14-13 所示。



图14-13　【安装应用程序】对话框

（8）单击【确定】按钮，安装程序会在桌面添加快捷图标。双击该图标即可在桌面窗口中运行 Silverlight 程序，效果如图 14-14 所示。

（9）Silverlight 桌面应用程序的卸载与安装一样容易。只需在窗口中右击，选择【删除此应用程序】命令，然后在弹出的对话框中单击【是】按钮即可，如图 14-15 所示。



图14-14　Silverlight 桌面运行效果

图14-15　卸载 Silverlight 桌面应用程序

14.6 实验指导——实现一个简易时钟

DispatcherTimer 是 Silverlight 中十分有用的一个计时器对象。DispatcherTimer 的使用十分简单，只需要为 DispatcherTimer 设置一个间隔时间，然后创建 Tick 的事件处理即可。当使用 Start()方法来开始计时后，Tick 事件就会根据设置的间隔时间来执行事件处理中的代码。

下面就使用 DispatcherTimer 对象来实现一个简易的时钟功能，程序根据 DispatcherTimer 的间隔时间来显示当前的时间。

（1）新建一个 Silverlight 4 应用程序。

（2）在界面设计器上添加一个 TextBlock 控件用于显示时间。具体代码如下所示。

```
<Grid x:Name="LayoutRoot" Background="White">
    <!--背景-->
    <Rectangle Fill="Gold" Stroke="Black" StrokeThickness="3 RadiusX="5"
     RadiusY="5"/>
    <!--显示时间-->
    <TextBlock x:Name="tbkTimer" width="300" Height="50" FontSize="30"
     Foreground="Red"/>
</Grid>
```

（3）在 MainPage.xaml.cs 文件中添加对 System.Windows.Threading 命名空间的引用。

```
using System.Windows.Threading;
```

（4）对默认的构造函数进行修改，增加计时器的创建、事件设置和开始代码，如下所示。

```
public MainPage()
{
    InitializeComponent();
    //创建 DispatcherTimer
    DispatcherTimer timer = new DispatcherTimer();
    //设置间隔 1 秒
    timer.Interval = new TimeSpan(0, 0, 1);
    //创建事件处理
    timer.Tick += new EventHandler(timer Tick);
    //开始计时
    timer.Start();
}
```

（5）在 timer_Tick()事件处理函数中获取最新的时间并显示到 tbkTimer 控件。代码如下。

```
private void timer Tick(object sender, EventArgs e)
{
    //输出时间
```

```
    tbkTimer.Text = "当前时间:" + DateTime.Now.ToLongTimeString();
}
```

（6）经过上面几步操作，时钟就制作完成了，运行效果如图 14-16 所示。

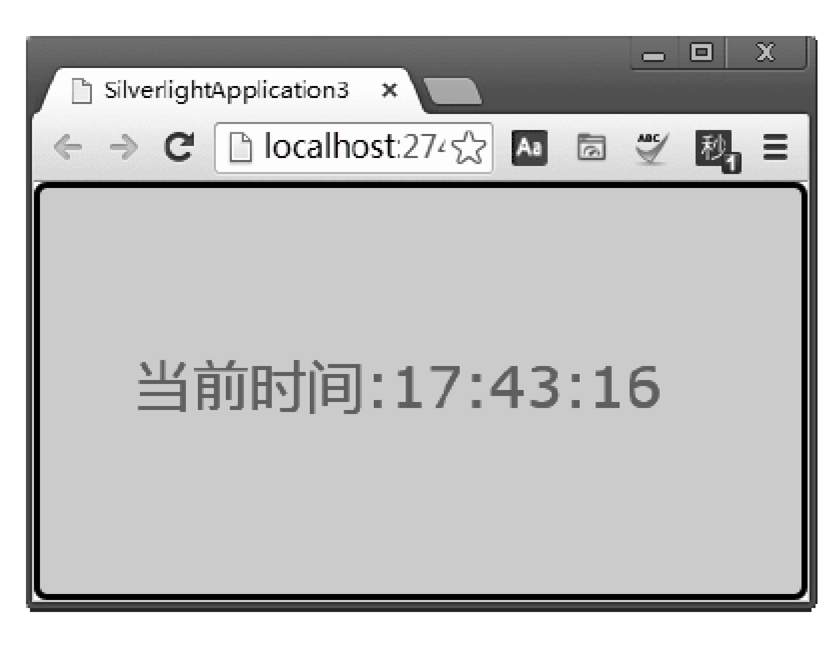


图14–16 时钟运行效果

14.7 实验指导——操作剪切板

Silverlight 4 增加了 System.Windows.Clipboard（剪切板操作）类，使用该类可以非常方便地操作剪切板中的数据。例如，让一些 Silverlight 应用程序中的文本可以复制到剪切板之中，同时也可以将从其他来源复制到剪切板中的内容粘贴到 Silverlight 应用程序之中，在此之前只能通过 JavaScript 来访问剪切板。

Clipboard 类支持 ContainsText()、SetText()和 GetText()三个方法，其中 ContainsText()可以返回一个 bool 类型的值，提示用户剪切板目前保存的类型是否是 Silverlight 所支持的 Unicode 类型字符，SetText()和 GetText()分别是用来设置和获取剪切板的文本数据。

下面通过一个案例讲解 Clipboard 类操作剪切板的方法。首先是在 Silverlight 应用程序中设计布局界面，假设在案例中包含两个 TextBox 控件和两个 Button 控件，具体代码如下。

```
<Grid x:Name="LayoutRoot" Width="400" Height="200" Background= "#FFD4
7F7F">
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition Width="100"/>
    </Grid.ColumnDefinitions>
    <TextBox x:Name="tbCopy" Width="260" Height="30"/>
    <TextBox x:Name="tbPaste" Width="260" Height="30" Grid.Row="1"/>
    <Button Content="复制" Grid.Column="1"  Click="btnCopy Click"
         x:Name="btnCopy" Width="80" Height="25"/>
    <Button Content="粘贴" Grid.Column="1" Grid.Row="1"  Click= "btn
     Paste_Click"
```

```
            x:Name="btnPaste" Width="80" Height="25"/>
</Grid>
```

上面的 btnCopy 按钮实现将 tbCopy 的内容复制到剪切板，btnPaste 按钮实现将剪切板的内容显示到 tbPaste。具体实现代码如下所示。

```
private void btnCopy_Click(object sender, RoutedEventArgs e)
{
    if (tbCopy.Text != string.Empty)
    {
        //设置剪切板
        Clipboard.SetText(tbCopy.Text);
    }
}
private void btnPaste Click(object sender, RoutedEventArgs e)
{
    //判断剪切板是否包括文本字符
    if (Clipboard.ContainsText())
    {
        //获取剪切板
        tbPaste.Text = Clipboard.GetText();
    }
}
```

运行程序，在第一次单击【复制】按钮时会弹出对话框询问是否允许 Silverlight 将内容复制到剪切板，在这里单击【是】按钮。之后单击【粘贴】按钮即可看到从剪切板中获取内容的效果，如图 14-17 所示。



图14–17 操作剪切板运行效果

思考与练习

一、填空题

1．Silverlight 应用程序编译器会创建一个__________文件。

2．在 XAML 中的所有元素都映射为一个.NET 类的实例。UserControl 元素表示在 WPF 中创建一个__________对象。

3．在 XAML 中使用__________属性来定义命名空间。

4．如果要在 Silverlight 中调用 HTML 必须

引用__________命名空间。

5．如果创建一个表示 HTML 页面的对象，应该使用__________代码。

6．DispatcherTimer 类所有的命名空间为__________。

二、选择题

1．Silverlight 是__________技术的一个子集。

A．WCF

B．WPF

C．XAML

D．ASP.NET

2．下列对 Silverlight 的描述，不正确的是__________。

A．可以跨浏览器运行

B．支持 TCP、WCF、SOAP 等网络技术

C．使用完整的.NET Framework 框架

D．没有 Menu 控件

3．在 XAML 中通过根元素的__________属性指定后台文件。

A．Class

B．CodeBehind

C．Background

D．CodeFile

4．Silverlight 应用程序的默认用户控件是__________。

A．MainPage.xaml

B．Default.xaml

C．Index.aspx

D．Index.xaml

5．如果一个方法使用了__________特性，那么可以在 HTML 中进行调用。

A．ScriptableMember

B．ClientMethod

C．STAthread

D．Serialization

6．下列不属于 Clipboard 类方法的是__________。

A．ContainsText()

B．SetText()

C．GetText()

D．IsContainsData()

三、简答题

1．简述什么是 Silverlight，以及与 WPF 的关系。

2．举例说明 Silverlight 应用程序的创建过程。

3．罗列三个以上 Silverlight 应用程序生成的文件。

4.简述 XAML 标记在 Silverlight 中的作用。

5．简述 Silverlight 与外部交互的方法。

第 15 章　ASP.NET MVC 4 框架

ASP.NET MVC 是一个全新的 Web 开发框架，它构建于.NET 平台之上，用于搭建松耦合、高维护性的应用程序。目前最新版本为 ASP.NET MVC 4，也是本章中采用的版本。通过本章的学习，读者可以在 VS 2012 中轻松地创建 ASP.NET MVC 4 应用程序、查看目录结构和搭建使用数据库的 MVC 应用程序。

此外，本章还准备了一些基础知识，对学习 MVC 时的重要概念进行讲解，帮助读者快速入门，如 MVC 工作模式、MVC 4 的 Razor 视图引擎、MVC 4 应用程序的运行流程等。限于篇幅关系，在本章不会对 ASP.NET MVC 4 的每个方面进行详细解释，有兴趣的读者可以查阅相关书籍。

本章学习要点：

- ❑ 了解 MVC 的工作模式及其优缺点
- ❑ 了解 ASP.NET MVC 4 的新增特性
- ❑ 熟悉 Razor 视图引擎的语法
- ❑ 掌握创建一个 ASP.NET MVC 4 应用程序的方法
- ❑ 熟悉 MVC 4 应用程序的组成元素
- ❑ 理解 MVC 4 应用程序的运行流程
- ❑ 掌握带数据库 MVC 4 应用程序的创建
- ❑ 熟悉控制器的添加
- ❑ 熟悉视图的使用

15.1 ASP.NET MVC 概述

ASP.NET MVC 是一种构建 Web 应用程序的框架，它将最流行的 MVC 框架应用于 ASP.NET 框架。下面从 MVC 的工作模式开始介绍，逐步过渡到 ASP.NET MVC 4。

15.1.1 MVC 工作模式

抛开具体的开发语言来讲，MVC 是一种软件开发的架构模式，也算是一种思想。在 MVC 模式中将软件系统分为：模型、视图和控制器三个基本部分。

它们各负其职，相互独立，又具有联系，如图 15-1 所示是 MVC 这三部分的示意图。

（1）模型：负责对整个软件项目数据和业务的封装和管理。

（2）视图：负责给用户传递信息，收集用户提交的信息。

（3）控制器：负责控制视图的展示逻辑。

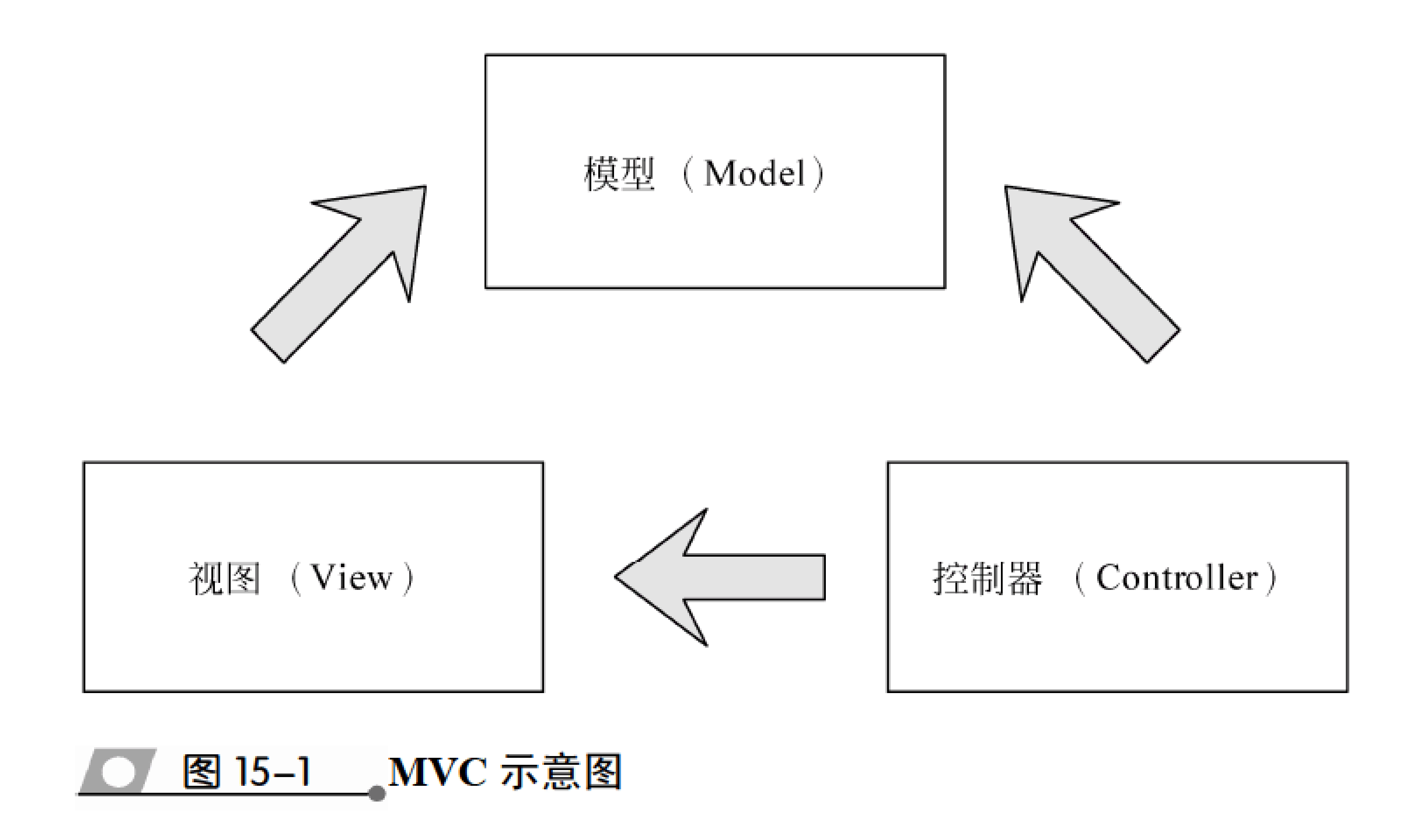


图 15-1 MVC 示意图

1. 视图

视图是用户看到并与之交互的界面。对老式的Web应用程序来说，视图就是由HTML元素组成的界面，在新式的 Web 应用程序中，HTML 依旧在视图中扮演着重要的角色。但一些新的技术已层出不穷，包括 Flash、XHTML、XML/XSL、WML 等一些标识语言和 Web Services。

如何处理应用程序的界面变得越来越有挑战性。MVC 一个很大的好处是它能为应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。

2. 模型

模型表示企业数据和业务规则。在 MVC 的三个部分中，模型拥有最多的处理任务。例如，它可能用 EJB 这样的组件对象来处理数据库。被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

3. 控制器

控制器接收用户的输入并调用模型和视图去完成用户的需求。所以，当单击 Web 页面中的超链接和发送 HTML 表单时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型组件去处理请求，然后确定用哪个视图来显示模型处理返回的数据。

15.1.2 MVC 优缺点

在使用 ASP 或者 PHP 开发 Web 应用时，初始的开发模板就是混合层的数据编程。例如，直接向数据库发送请求并用 HTML 显示，开发速度往往比较快。但由于数据页面

的分离不是很直接，因而很难体现出业务模型的样子或者模型的重用性，很难满足用户的变化性需求。

MVC 要求对应用分层，虽然要花费额外的工作，但产品的结构清晰，产品的应用通过模型可以得到更好的体现。

（1）首先，最重要的是应该有多个视图对应一个模型的能力。在目前用户需求的快速变化下，可能有多种方式访问应用的要求。

例如，订单模型可能有本系统的订单，也有网上订单，或者其他系统的订单，但对于订单的处理都是一样，也就是说订单的处理是一致的。按 MVC 设计模式，一个订单模型以及多个视图即可解决问题。这样减少了代码的复制，即减少了代码的维护量，一旦模型发生改变，也易于维护。

（2）其次，由于模型返回的数据不带任何显示格式，因而这些模型也可直接应用于接口的使用。

（3）再次，由于一个应用被分离为三层，因此有时改变其中的一层就能满足应用的改变。一个应用的业务流程或者业务规则的改变只需改动 MVC 的模型层。

（4）控制层的概念也很有效，由于它把不同的模型和不同的视图组合在一起完成不同的请求，因此，控制层可以说是包含用户请求权限的概念。

（5）最后，它还有利于软件工程化管理。由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化产生管理程序代码。

凡事都不是绝对的，MVC 也是如此。MVC 也不是最先进、最优秀或者最好的选择，其缺点主要体现在以下几个方面：

（1）增加了系统结构和实现的复杂性。

对于简单的界面，严格遵循 MVC，使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。

（2）视图与控制器间过于紧密的连接。

视图与控制器是相互分离的，但是又紧密联系的部分，如果视图没有控制器的存在，那它的应用是很有限的。反之亦然，这样就妨碍了它们的独立重用。

（3）视图对于模型数据的低效率访问。

依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。

目前，一般高级的界面工具或者构造器不支持 MVC 架构。改造这些工具以适应 MVC 需要和建立分离部分的代价是很高的，从而造成使用 MVC 的困难。

15.1.3 ASP.NET MVC 4 新特性

ASP.NET MVC 是微软官方推出的基于 ASP.NET 的 MVC 模式网站应用程序开发框架，官方网站 http://www.asp.net/mvc。

ASP.NET MVC 的第一个版本是于 2009 年 3 月 17 日发布的 RTM 版本。在经历了漫

长的 Preview 之后，终于 2010 年 3 月 11 日发布了 ASP.NET MVC 2.0。之后又发布了 ASP.NET MVC 3.0。目前最新版本为 MVC 4.0，也是本书所采用的开发版本。

ASP.NET MVC 4 包含大量的新特性和功能，主要新特性如下：

（1）Razor 增强功能

ASP.NET MVC 4 提供了新的视图引擎 Razor V2。Razor V2 包含丰富的增强功能，可以让视图模板更简洁，并为解析 URL 引用和有选择地呈现 HTML 属性提供了更好的支持。

（2）捆绑和缩小

ASP.NET MVC 4 包括 ASP.NET 4.5 中的新捆绑和缩小支持特性。这些特性使用户在构建 Web 应用程序时能最小化网页所做的 HTTP 请求数量和大小，实现更快的加载速度和响应用户请求。

（3）数据库迁移

ASP.NET MVC 4 内置的实体框架为最新的 4.3，包含很多新功能，其中最受期待的功能之一是支持迁移数据库。这让用户方便地使用代码集中迁移的方法来改进数据库架构，这样做的同时也保留了数据库内的数据。

（4）Web API

ASP.NET MVC 4 包括一些出色的新特性支持来创建 Web API。这使用户能够轻松地创建 HTTP 服务和应用程序，并且这些服务和应用程序能够以编程方式从大范围的客户端调用（包括使用 JavaScript 从浏览器中调用，到任何移动/客户端平台上的本机应用程序）。新的 Web API 还支持提供一个理想的平台来建设 RESTFUL 服务。

（5）移动 Web

ASP.NET MVC 4 支持用于构建移动 Web 应用程序和移动 Web 站点，并使其更容易构建针对手机和平板的优化体验。

（6）异步支持和 WebSockets

在将 ASP. NET MVC 4 与.NET 4.5 和 VS 2012 一起使用时，用户将能够充分利用额外的语言和运行时功能，异步支持是其中最大的一个。ASP.NET MVC 运行时支持新的 C#异步语言增强功能，这将让用户编写出令人难以置信的可伸缩的应用程序。同时还可以利用内置于.NET 4.5 的新 WebSocket 支持来构建更丰富的浏览/服务器通信应用程序。

15.1.4 Razor 视图引擎

在 ASP.NET MVC 1 和 ASP.NET MVC 2 中默认使用的视图引擎被称为 Web Forms 视图引擎。因为它与 Web Forms 使用了相同的文件（例如，aspx、ascx 和 master）和语法。

如下所示为一个 Web Forms 视图引擎下的文件内容。

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
```

```
Inherits="System.Web.Mvc.ViewPage<MvcApplication1.Models.LogOnModel>" %>
<asp:Content ID="loginTitle" ContentPlaceHolderID="TitleContent" runat=
"server">
    登录
</asp:Content>
<asp:Content ID="loginContent" ContentPlaceHolderID="MainContent"
runat="server">
    <h2>登录</h2>
    <p>请输入用户名和密码。 <%: Html.ActionLink("注册", "Register") %> 如果
    您没有账户。</p>
    <script src="<%: Url.Content("~/Scripts/jquery.validate.min.js") %>"
    type="text/javascript"></script>
    <% using (Html.BeginForm()) { %>
        <%: Html.ValidationSummary(true, "登录不成功。请更正错误并重试。") %>
        <div>
            <fieldset>
                <legend>账户信息</legend>
                <div class="editor-label">
                    <%: Html.LabelFor(m => m.UserName) %>
                </div>
                <div class="editor-field">
                    <%: Html.TextBoxFor(m => m.UserName) %>
                    <%: Html.ValidationMessageFor(m => m.UserName) %>
                </div>
                <div class="editor-label">
                    <%: Html.LabelFor(m => m.Password) %>
                </div>
                <div class="editor-field">
                    <%: Html.PasswordFor(m => m.Password) %>
                    <%: Html.ValidationMessageFor(m => m.Password) %>
                </div>
                <div class="editor-label">
                    <%: Html.CheckBoxFor(m => m.RememberMe) %>
                    <%: Html.LabelFor(m => m.RememberMe) %>
                </div>
                <p><input type="submit" value="登录" /> </p>
            </fieldset>
        </div>
    <% } %>
</asp:Content>
```

从 ASP.NET MVC 2 之后新增了 Razor 视图引擎，它不像 Web Forms 视图引擎那样具有类似 XML 的复杂语法规则。Razor 的语法更加易于输入、阅读和维护，且结构更简洁。如下所示是使用 Razor 生成相同标记的内容。

```
@model MvcApplication2.Models.LogOnModel
@{
    ViewBag.Title = "登录";
}
<h2>登录</h2>
<p>请输入用户名和密码。如果您没有账户，请 @Html.ActionLink("注册", "Register")。
</p>
<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type=
"text/javascript"></script>
@Html.ValidationSummary(true, "登录不成功。请更正错误并重试。")
@using (Html.BeginForm()) {
    <div>
        <fieldset>
            <legend>账户信息</legend>
            <div class="editor-label">  @Html.LabelFor(m => m.UserName)
            </div>
            <div class="editor-field">
                @Html.TextBoxFor(m => m.UserName)
                @Html.ValidationMessageFor(m => m.UserName)
            </div>
            <div class="editor-label">
                @Html.LabelFor(m => m.Password)
            </div>
            <div class="editor-field">
                @Html.PasswordFor(m => m.Password)
                @Html.ValidationMessageFor(m => m.Password)
            </div>
            <div class="editor-label">
                @Html.CheckBoxFor(m => m.RememberMe)
                @Html.LabelFor(m => m.RememberMe)
            </div>
            <p> <input type="submit" value="登录" />  </p>
        </fieldset>
    </div>
}
```

Razor 视图引擎允许用户编写更复杂的逻辑代码，而且可以在代码和标签之间轻易转换。虽然 Razor 语法与其他的标记语法不同（ASP.NET Web Form），但是它们的目标都是相同的，即渲染 HTML。

如下是 foreach 循环在 Web Form 中的语法。

```
<ul>
<% foreach(var a in actions) {%>
<li><a href="<%:a.href%>" > <%: a.title%></a> </li>
<%}%>
```

```
</ul>
```

如下所示为 Razor 语法的代码。

```
<ul>
@foreach(var a in actions){
    <li><a href="@a.href">@a.title</a></li>
}
</ul>
```

在 Razor 中要定义一个代码块需要以“@{”开始，以“}”结束。这里的代码块与代码段不同，代码块要求使用 C#常规代码，即必须符合 C#语法，每行以分号结束。如下是一个代码块的例子。

```
@{
    LayoutPage="~/Views/Shared/ Layout.cshtml";
    View.Title="Action:"+Model.Title;
}
```

如上所示，代码块中不能渲染任何东西，并且编写的代码不可以有返回值。

综上所述，Razor 的主要特点如下。

（1）不是新语言。Razor 语法非常直观，且支持现在的.NET 编码方法和技术。只需简单地输入“@”符号即可进入 Razor 视图状态。

（2）容易学习。由于 Razor 不是一门新的编程语言，因此学习 Razor 非常容易。如果熟悉 HTML 和.NET 语言，那么编写将会更加容易上手 Razor。

（3）支持所有文本编辑器。由于 Razor 是轻量级并注重于 HTML 的语言，因此可以自由地选择编辑器。虽然 VS 2012 提供了 Razor 的智能语法提示，但是 Razor 足够简单，可以使用任何文本编辑器进行编辑。

（4）单元测试。Razor 视图引擎的核心编译与 System.Web 或 ASP.NET 没有任何依赖关系，它能执行单元测试，甚至可以从命令行执行。

15.2 实验指导——创建第一个 MVC 4 项目

15.1 节介绍了 MVC 的工作模式，ASP.NET MVC 4 的新特性以及 Razor 的简介。本节通过创建第一个 MVC 项目演示 MVC 的基本操作。具体步骤如下。

（1）打开 VS 2012，选择【文件】|【新建】|【项目】命令打开【新建项目】对话框。

（2）从左侧选择 Web 分类模板，之后右侧会出现很多 Web 项目列表。要创建 MVC，所以选择【ASP.NET MVC 4 Web 应用程序】选项。

（3）在下方的【名称】文本框中为要创建的 MVC 项目起一个有意义的名称，这里为 MvcApplication，如图 15-2 所示。

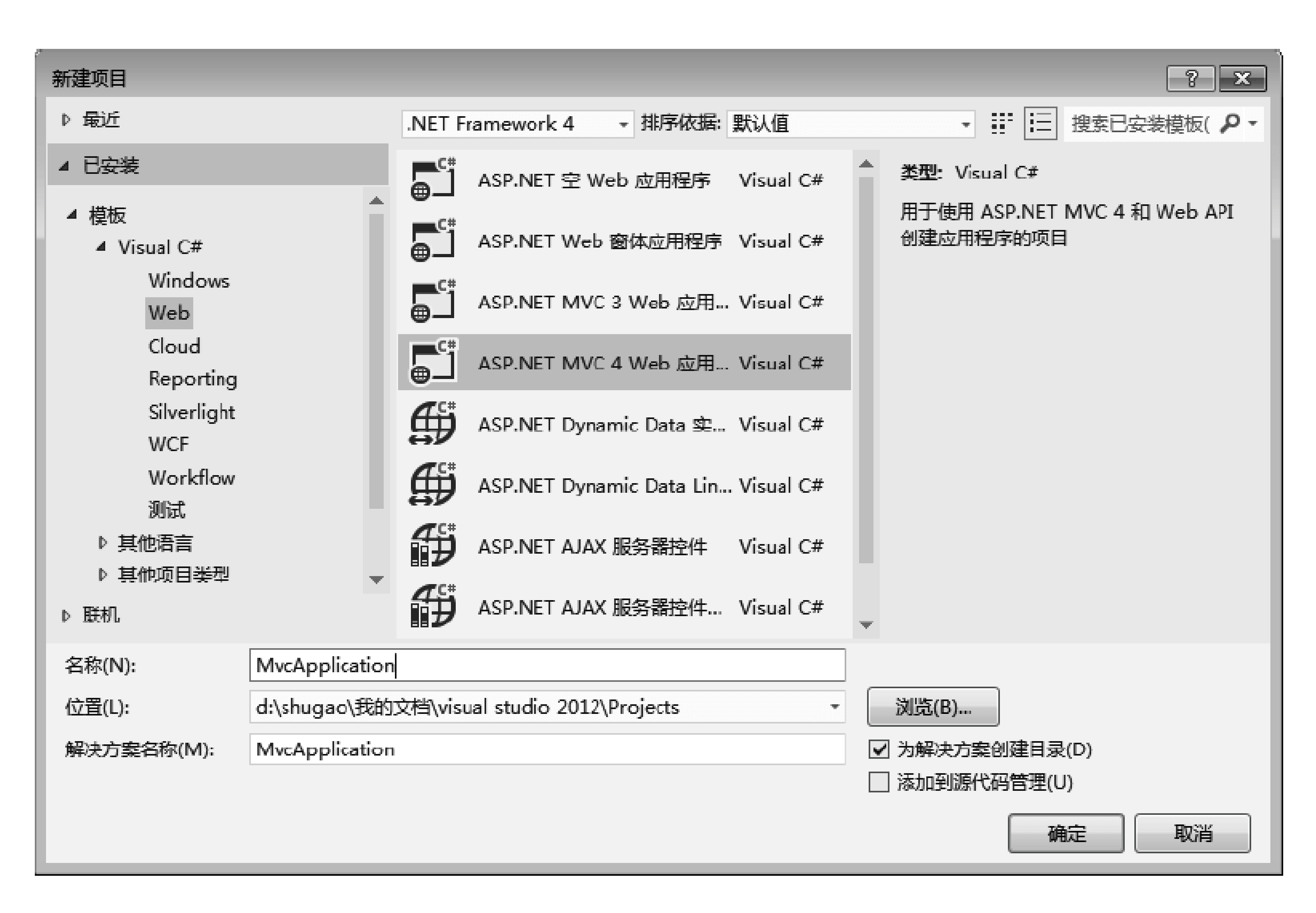


图 15-2 【新建项目】对话框

（4）单击【确定】按钮之后会进入 MVC 项目模板设计界面。在这里可以选择 MVC 项目使用的模板、视图引擎以及是否创建对应的单元测试项目，如图 15-3 所示。

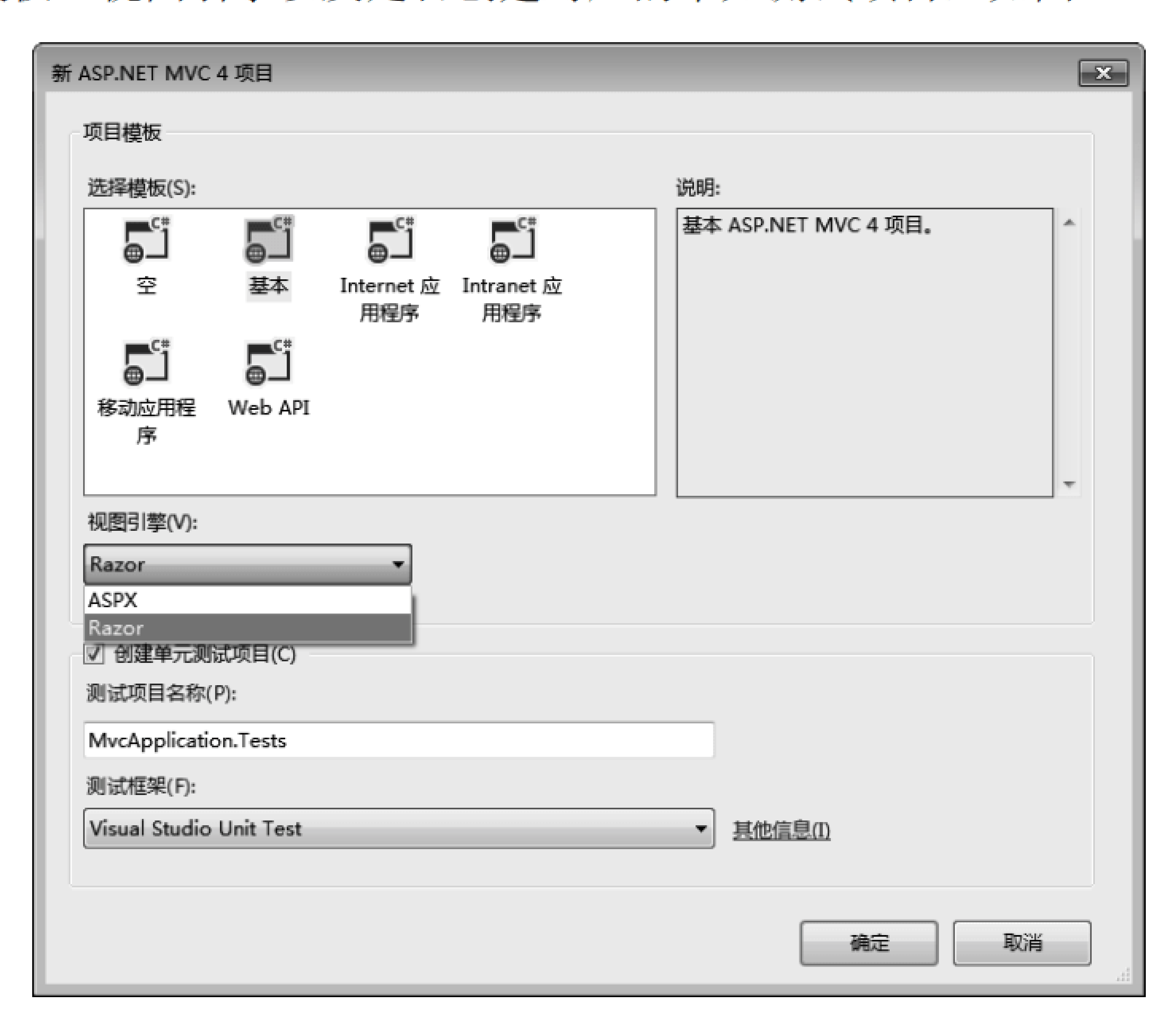


图 15-3 设置 MVC 项目

在图 15-3 中可以设置 6 种 MVC 项目模板，每种的含义如下。

① 空模板

空模板用于创建 ASP.NET MVC 4 网站的架构，包含基本的文本夹结构，以及需要引用的 ASP.NET MVC 程序集，也包含可能要使用的 JavaScript 库。该模板同样包含默认的视图内存卡，以及标准配置代码的 Global.asax 文件。

② 基本模板

基本模板会按照 ASP.NET MVC 4 规则创建文件夹结构，包含 ASP.NET MVC 程序集的引用。该模板表明了创建 ASP.NET MVC 4 项目所需要的最低标准资源。通常情况下选择此模板，本示例中也是选择该模板。

③ Internet 应用程序模板

Internet 应用程序模板是对空模板的扩展，包含简单的默认控制器、账户控制器。账户控制器包含用户注册和登录网站所需要的基本逻辑代码，以及这两个控制器需要的默认视图文件。

④ Intranet 应用程序模板

Intranet 应用程序模板与 Internet 应用程序模板类似，都是对空模板的扩展。不同的是 Intranet 应用程序模板使用了基于 Windows 的验证机制，这也是企业局域网安全验证的首要机制。

⑤ 移动应用程序模板

移动应用程序模板是 Internet 应用程序模板针对移动设备的优化版本，它包含 jQuery Mobile JavaScript 框架以及与 jQuery Mobile 完美兼容的视图模板。

⑥ Web API 模板

Web API 模板是 Internet 应用程序模板的扩展，它预定义了 Web API 控制器。Web API 是一种新的、轻量级的 RESTfull HTTP Web 服务框架，可以与 ASP.NET MVC 无缝集成。该模板是创建 Ajax 交互数据服务的首选模板。

（5）在如图 15-3 所示的界面中将视图引擎设置为 Razor。

（6）如果启用【创建单元测试项目】复选框还需要为测试项目指定一个名称，这里不启用。

（7）以上选项设置完成之后单击【确定】按钮。此时，VS 2012 将会在指定位置创建 MVC 项目，并复制目录和文件，以及进行预定义的配置工作。

（8）至此，一个最简单的 MVC 4 项目就算创建完成了。运行项目，将在浏览器中看到项目的运行效果，如图 15-4 所示。



图 15-4 默认运行效果

15.3 MVC 4 项目元素详解

15.2 节创建了第一个 MVC 4 项目，并在浏览器中进行了测试。在创建过程中会有很多选项，同时 VS 2012 会生成相应的框架和文件。本节以前面创建的 MVC 项目为例，详细介绍 MVC 4 项目的重要组成元素。

15.3.1 MVC 4 应用程序目录结构

在使用 VS 2014 创建一个使用基本模板的 MVC 4 应用程序之后，将会自动向这个应用程序中生成很多文件和目录。如图 15-5 所示为默认的目录结构。

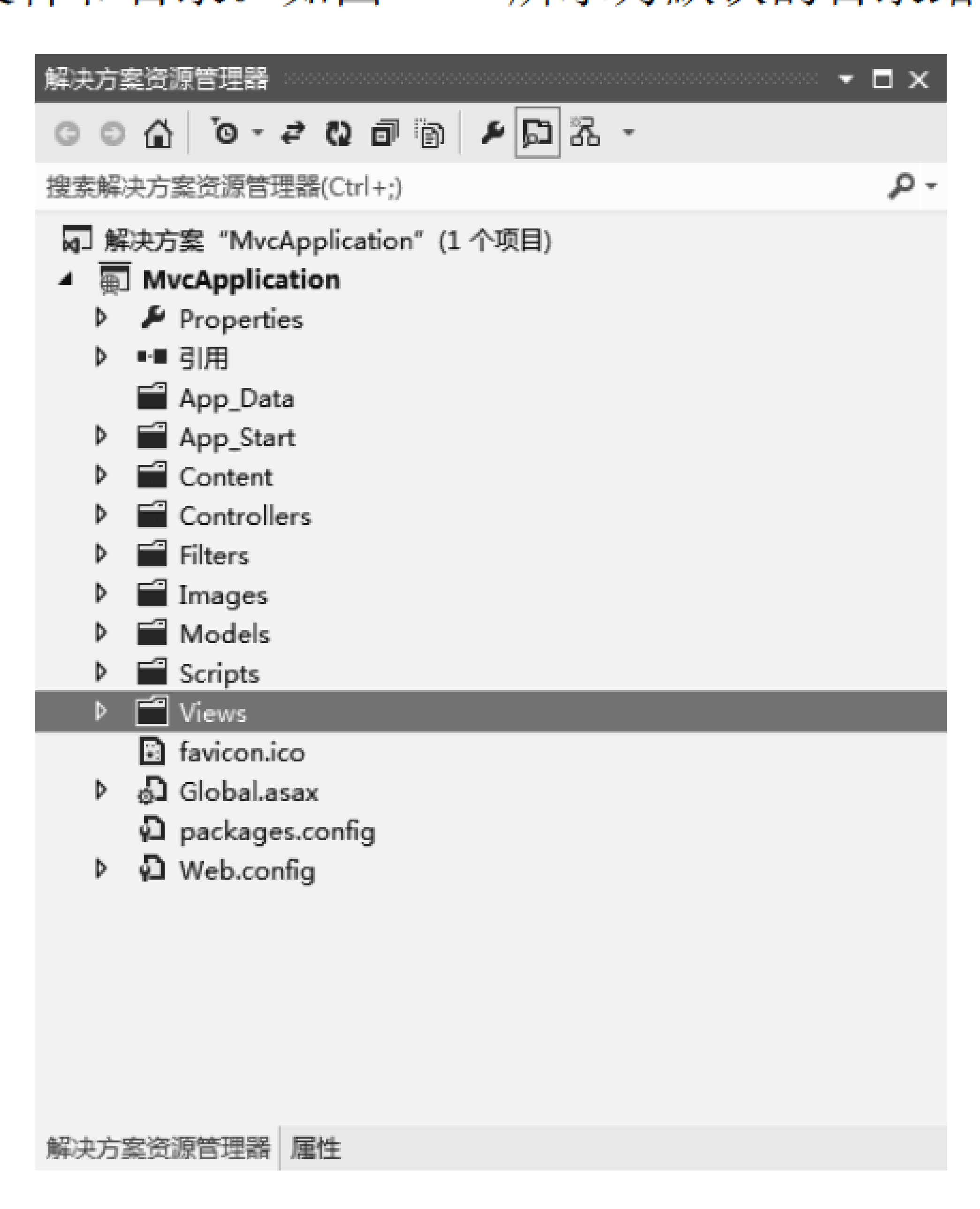


图 15-5 MVC 应用程序目录结构

默认生成 9 个顶级目录，具体说明如表 15-1 所示。

表 15-1 MVC 应用程序目录及说明

目录名称	说明
Controllers	该目录用于保存处理 URL 请求的 Controller 类
Models	该目录用于保存表示和操作数据以及业务对象的类
Views	该目录用于保存负责显示输出结果的视图文件
Scripts	该目录用于保存 JavaScript 库文件和脚本
Images	该目录用于保存图像文件
Content	该目录用于保存除了脚本、样式表和图片之外的其他内容
Filters	该目录用于保存过滤器文件
App_Data	该目录用于保存数据库和数据文件
App_Start	该目录用于保存应用程序用到的配置文件后台代码

如表 15-1 所示，默认的目录结构提供了一个很好的目录约定，使得应用程序文件的管理非常清晰。但是 ASP.NET MVC 4 并不强制要求使用上面那些目录结构。例如，在大型应用程序中开发人员通常把数据和业务逻辑等放到单独的一个项目中。

15.3.2 MVC 4 的约定优于配置

在默认情况下，ASP.NET MVC 4 应用程序对约定的依赖性很强。这样就避免了开发人员配置具体化约定可以设置的项。例如，当解析视图模板时，ASP.NET MVC 4 采用一种基于约定的目录命名结构。这个约定可以使在 Controller 类中引用视图引擎时省略位置路径信息。默认情况下，ASP.NET MVC 4 会在应用程序\Views\控制器名称\目录下查找视图模板文件。

所谓约定优于配置是指“已经掌握如何创建一个 Web 应用程序，现在将以往积累的经验应用于框架中，从而避免开发时针对每一项的配置工作。”

在 ASP.NET MVC 4 的如下三个核心目录中就使用了约定优于配置规则。

（1）Controllers 目录。

（2）Models 目录。

（3）Views 目录。

根据 MVC 4 的目录结构以及约定优于配置原则，可以很容易地预测到程序结构：

（1）每一个控制器类的名称都以 Controller 结束，例如 ProductController、HomeController 等，这些控制器类都存放在 Controllers 目录中。

（2）应用程序中的所有视图都存放在 Views 下的一个单独目录中。

（3）控制器使用的视图是在 Views 主目录的一个子目录中，该子目录是根据控制器名称来命名的。例如，Product 控制器使用的视图放在/Views/Product 目录中。

（4）所有可重用的 UI 元素都位于一个相似的结构中，共享目录 Views/Shared 用于存放其他视图。

15.3.3 MVC 4 项目中的模型、视图与控制器

在构建传统的 ASP.NET Web Forms 应用程序时，URL 和页面是一一对应的。如果从服务器上请求名称为 Index.aspx 的页面，则对应网站目录下名为 Inex.aspx 的文件。如果 Index.aspx 文件不存在，则将出现“404 – Page Not Found”错误。

相反，在构建 ASP.NET MVC 4 应用程序时，在浏览器地址栏中输入的 URL 和应用程序中的文件不存在对应关系。在 ASP.NET MVC 应用程序中，URL 对应的是控制器操作，而不是网站下的文件。

还有，在传统的 ASP.NET 应用程序中，浏览器请求是映射到页面上的。在 ASP.NET MVC 应用程序中，浏览器请求是映射到控制器操作的。ASP.NET Web Forms 应用程序

关注的是内容，而 ASP.NET MVC 应用程序关注的是应用程序逻辑。

1. 控制器

控制器负责控制用户与 MVC 应用程序的交互方式。控制器决定在用户发出浏览器请求时向用户发送什么样的响应。

控制器只是一个类。ASP.NET MVC 示例应用程序包括一个名称为 HomeController.cs 的控制器，该控制器位于 Controllers 文件夹内。

HomeController.cs 的内容如下所示。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace MvcApplication.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewBag.Message = "修改此模板以快速启动你的 ASP.NET MVC 应用程序。";
            return View();
        }
        public ActionResult About()
        {
            ViewBag.Message = "你的应用程序说明页。";
            return View();
        }
        public ActionResult Contact()
        {
            ViewBag.Message = "你的联系方式页。";
            return View();
        }
    }
}
```


从以上代码中看到，HomeController 有三个方法，名称分别为 Index()、About()和 Contact()。这三个方法对应于控制器公开的三个操作。URL“/Home/Index”调用 HomeController.Index()方法，URL“/Home/About”调用 HomeController.About()方法，而 URL“/Home/Contact”调用 HomeController.Contact()方法。

控制器中的任何公共方法都作为控制器操作被公开。这意味着通过向浏览器输入正确的 URL 来访问 Internet 的任何人都可以激活包含在控制器中的任何公共方法。

2. 视图

HomeController 类的三个公共控制器方法都返回了一个视图。视图包括发送到浏览器的 HTML 标记和内容。在使用 ASP.NET MVC 应用程序时，视图就等于页面。

因此，必须在正确的位置创建视图。HomeController.Index()操作返回位于以下路径的视图。

```
\Views\Home\Index.aspx
```

HomeController.About()操作返回位于以下路径的视图。

```
\Views\Home\About.aspx
```

总之，如果要为控制器操作返回视图，则需要在 Views 文件夹中使用与控制器相同的名称创建子文件夹。在子文件夹中，必须创建与控制器操作名称相同的.aspx 文件。

如下所示为 About.aspx 视图文件中所包含的代码。

```
@{
    ViewBag.Title = "关于";
}
<hgroup class="title">
    <h1>@ViewBag.Title.</h1>
    <h2>@ViewBag.Message</h2>
</hgroup>
<article>
    <p>使用此区域可提供附加信息。</p>
    <p>使用此区域可提供附加信息。</p>
    <p>使用此区域可提供附加信息。</p>
</article>
<aside>
    <h3>副标题</h3>
    <p> 使用此区域可提供附加信息。 </p>
    <ul>
        <li>@Html.ActionLink("主页", "Index", "Home")</li>
        <li>@Html.ActionLink("关于", "About", "Home")</li>
        <li>@Html.ActionLink("联系方式", "Contact", "Home")</li>
    </ul>
</aside>
```

如上述代码所示，视图文件由 Razor 标记和标准 HTML 组成。在此处可以输入任何想要的内容来修改视图的显示效果。

提 示

视图非常类似于 ASP.NET Web Forms 的页面。视图包括 HTML 内容和脚本。可以使用熟悉的 C#语言编写脚本，使用脚本显示动态内容。

3. 模型

前面已经讨论控制器和视图，最后讨论一下模型。MVC 模型包含所有视图或控制器不包含的应用程序逻辑。模型应该包含所有应用程序业务逻辑和数据库访问逻辑。例如，如果正在使用 LINQ to SQL 访问数据库，那么将在 Models 文件夹中创建 LINQ to SQL 类（dbml 文件）。

视图应该只包含与生成用户界面相关的逻辑。控制器应该只包含要求返回正确视图或将用户重定向到另一操作所需的最小逻辑。其他所有内容都应包含在模型中。

总之，应该努力实现高效模型和简化控制器。控制器方法应该只包含几行代码。如果控制器操作过长，则应该考虑将逻辑移动到 Models 文件夹中的一个新类中。

15.3.4 MVC 4 路由规则

除了使用前面列出的文件夹之外，ASP.NET MVC Web 应用程序还使用 Global.asax 文件中的代码来设置全局 URL 路由默认值。

路由在 Global.asax 文件的 Application_Start()方法中初始化。下面的示例演示一个普通 Global.asax 文件。

```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application Start()
    {
        AreaRegistration.RegisterAllAreas();
        WebApiConfig.Register(GlobalConfiguration.Configuration);
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);  //注册路由
        BundleConfig.RegisterBundles(BundleTable.Bundles);
        AuthConfig.RegisterAuth();
    }
}
```

Application_Start()方法在应用程序启动时被调用，在方法内对应用程序所需的各种配置信息进行注册。其中路由规则保存在 App_Start 目录下的 RoteConfig.cs 文件中，由 RouteConfig.RegisterRoutes(RouteTable.Routes)语句来初始化。

RoteConfig 类中包含默认路由规则的代码如下。

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}"); ❶
        routes.MapRoute( ❷
            name: "Default",                    //路由名称
```

```
                url: "{controller}/{action}/{id}",   //带有参数的 URL
                defaults: new { controller = "Home", action = "Index", id =
                UrlParameter.Optional } //参数默认值
            );
        }
    }
```

在上述代码中定义了两个 URL 路由。在❶处定义了可以忽略的路由配置，也就是说，不需要路由处理程序去处理这些路由；而在❷处则设置了一个默认的路由。

当 ASP.NET MVC 应用程序第一次启动时，调用 Application_Start()方法。该方法又调用 RegisterRoutes()和 RegisterRoutes()方法创建默认的路由表。

默认的路由表中包含一个路由。该默认路由将所有进入的请求拆分为三个单元(URL 单元是正斜杠之间的所有内容)。第一个单元映射到控制器名称，第二个单元映射到操作名称，最后一个单元映射到传递给操作名称 ID 的参数。

例如，考虑下面的 URL：

```
/Product/Details/3
```

此 URL 被解析为如下三个部分。

```
Controller = ProductController
Action = Details
Id = 3
```



前缀控制器将被附加到控制器参数的末端，这只是 MVC 的一个特殊之处。

默认路由包括所有三个单元的默认值。默认控制器是 HomeController，默认操作是 Index，而默认 ID 是一个空字符串。观察这些默认值，考虑如何解析下面的 URL。

```
/Employee
```

此 URL 被解析为如下三个参数。

```
Controller = EmployeeController
Action = Index
Id = ""
```

最后，如果打开 ASP.NET MVC 应用程序而不提供任何 URL(例如 http://localhost/)，那么 URL 将被解析为：

```
Controller = HomeController
Action = Index
Id = ""
```

请求将被发送到 HomeController 类上的 Index()操作。

15.4 ASP.NET MVC 4 应用程序运行流程

15.3 节对 ASP.NET MVC 4 项目的核心元素进行了详细介绍，本节介绍一下应用程序的运行流程，如图 15-6 所示。

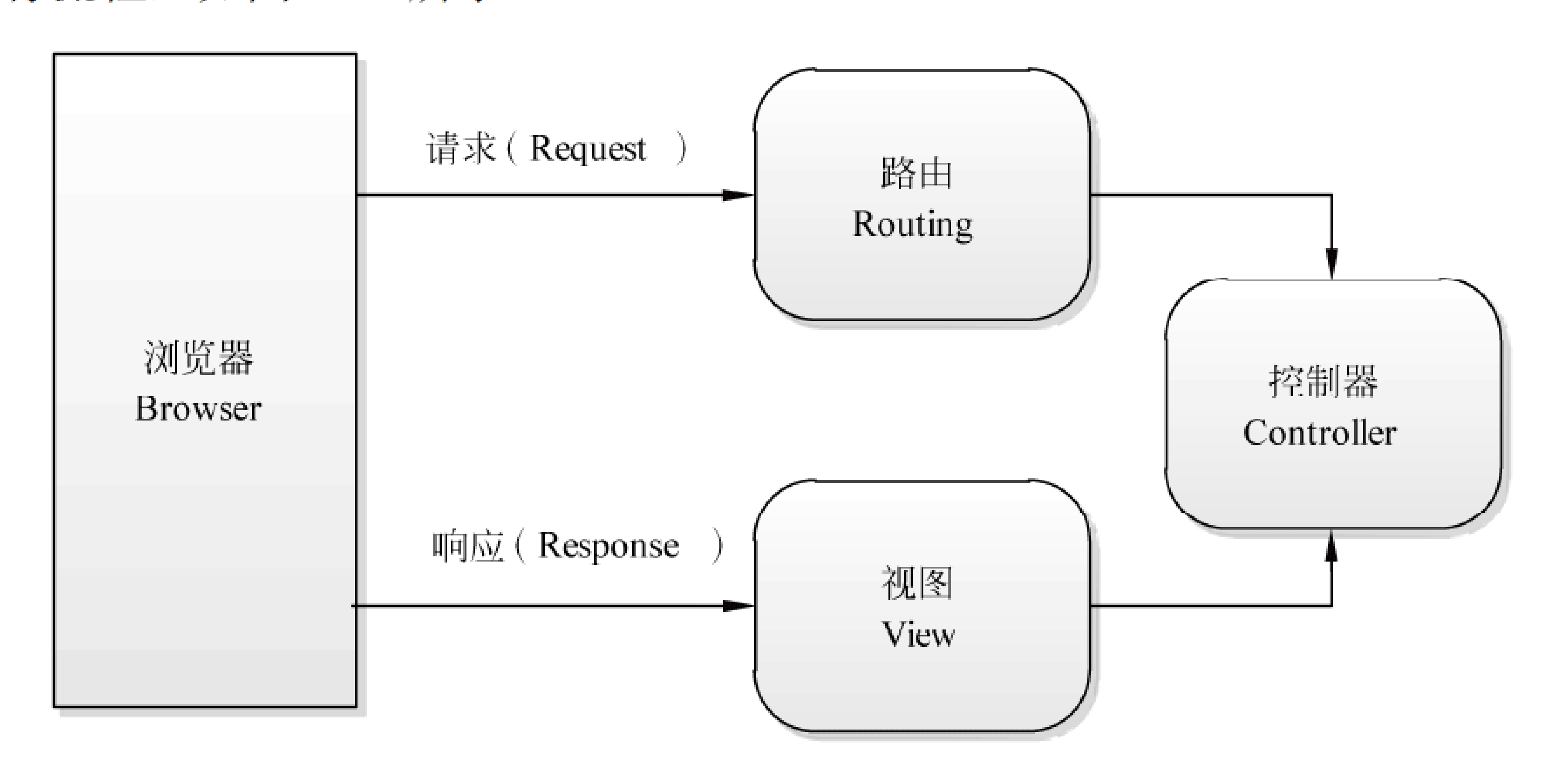


图 15-6 MVC 应用程序运行流程

在 ASP.NET MVC 应用程序的运行过程中请求会发送到 Controllers 中，这样就对应了 ASP.NET MVC 应用程序中的 Controllers 文件夹，Controllers 只负责数据的读取和页面逻辑的处理。

在 Controllers 读取数据时，需要通过 Models 从数据库中读取相应的信息，读取数据完毕后，Controllers 再将数据和 Controller 整合并提交到 Views 视图中，整合后的页面将通过浏览器呈现在用户面前。

当用户使用 URL“http://localhost:19518/Home/About”发送请求时，这个请求首先会发送到 Controllers 中，Controllers 通过 Global.asax 文件中的路由设置进行相应的 URL 映射，Global.asax 文件相应代码如下所示。

```
public static void RegisterRoutes(RouteCollection routes)
//注册路由
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        "Default",
        "{controller}/{action}/{id}",
        new { controller = "Home", action = "Index", id = UrlParameter.
        Optional }        //配置路由
    );
}
```

下面来看看 Controllers 文件夹内的文件，以及 Views 文件夹的文件。在 Views 中包

含 Home 文件夹，在 Home 文件夹中存在 About.cshtml 和 Index.cshtml 文件，而同样在 Controllers 文件夹中包含与 Home 文件夹同名的 HomeController.cs 文件。

当用户访问 http://localhost:19518/Home/About 路径时，该路径请求会传送到 Controller 中。

注意

在 Controllers 文件夹中创建 HomeController.cs 文件同 Home 是同名文件，在 Controllers 中创建的文件，其文件名后的 Controller.cs 是不能更改的，所以 HomeController.cs 文件也可以看作是 Home 文件夹的同名文件。

在 Controller 中，Controller 通过 Global.asax 文件和相应的编程实现路径的映射，示例代码如下所示。

```
public ActionResult About()                    //实现 About 页面
{
    ViewBag.Message = "你的应用程序说明页。";
    return View();                             //返回视图
}
```

上述代码实现了 About 的页面呈现，在运行相应的方法后会返回一个 View。这里默认返回的是与 Home 的 About()方法同名的页面，即 About.cshtml。

将 About.cshtml 页面中的文字进行相应的更改，修改后代码如下所示。

```
@{
    ViewBag.Title = "关于";
}
<hgroup class="title">
    <h1>@ViewBag.Title.</h1>    <h2>@ViewBag.Message</h2></hgroup>
<article>
    <p>这是 Home 控制器 About()方法返回的页面</p>
</article>
```

修改后 About.cshtml 页面运行结果如图 15-7 所示。



图 15-7 About.cshtml 页面

从上述代码可以看出，Controllers 与 Global.asax 用于 URL 的映射，而 Views 用于页面的呈现。从这里可以看出，当用户访问 http://localhost:19518/Home/About 页面时，访问的并不是服务器中的/Home/About 页面，而访问的是 Controllers 中的 HomeControllers 的 About()方法，而 About()方法又调用视图文件 About.cshtml 作为呈现。

> **提 示**
>
> ASP.NET MVC 应用程序中的 URL 路径访问的并不是一个页面，而是一个方法，例如，访问/Home/About 页面就是访问的是 HomeControllers 中的 About()方法，而访问/Account/Login 页面就是访问的是 AccountControllers 中的 Login()方法。

在这个默认创建的 ASP.NET MVC 示例应用程序中，对应关系如图 15-8 所示。

```
HomeController.cs ──────────────────────────── Home

public ActionResult Index() ─────────────────── Index.cshtml
{
    ViewBag.Message = "修改此模板以快速启动你的 ASP.NET MVC 应用程序。";
    return View();
}
public ActionResult About() ─────────────────── About.cshtml
{
    ViewBag.Message = "你的应用程序说明页。";
    return View();
}
public ActionResult Contact() ───────────────── Contact.cshtml
{
    ViewBag.Message = "你的联系方式页。";
    return View();
}
```

图 15-8　MVC 应用程序对应关系图

在 ASP.NET MVC 应用程序中，HomeController.cs 对应 Views 的 Home 文件夹，而其中的 Index()方法对应 About.cshtml 文件，About()方法对应 About.cshtml 文件，Contact()方法对应 Contact.cshtml 文件。

> **提 示**
>
> 在命名时，默认情况下 XXXController.cs 对应 Views 的 XXX 文件夹，而其中 XXXController.cs 中的 YYY()方法对应 XXX 文件夹中的 YYY.aspx，而访问路径为 XXX/YYY 是访问的是 XXXController.cs 中的 YYY()方法。

15.5 实验指导——管理图书信息

通过前面的讲解，相信读者对 MVC 4 应用程序的创建、其组成部分和运行流程有所

了解。本节将使用 MVC 4 完成一个针对图书信息的增、删、改和查功能。具体步骤如下。

（1）创建一个名为 BookMvcApp 的 MVC 4 应用程序，并使用空模板来生成默认框架。此时的应用程序目录结构如图 15-9 所示。

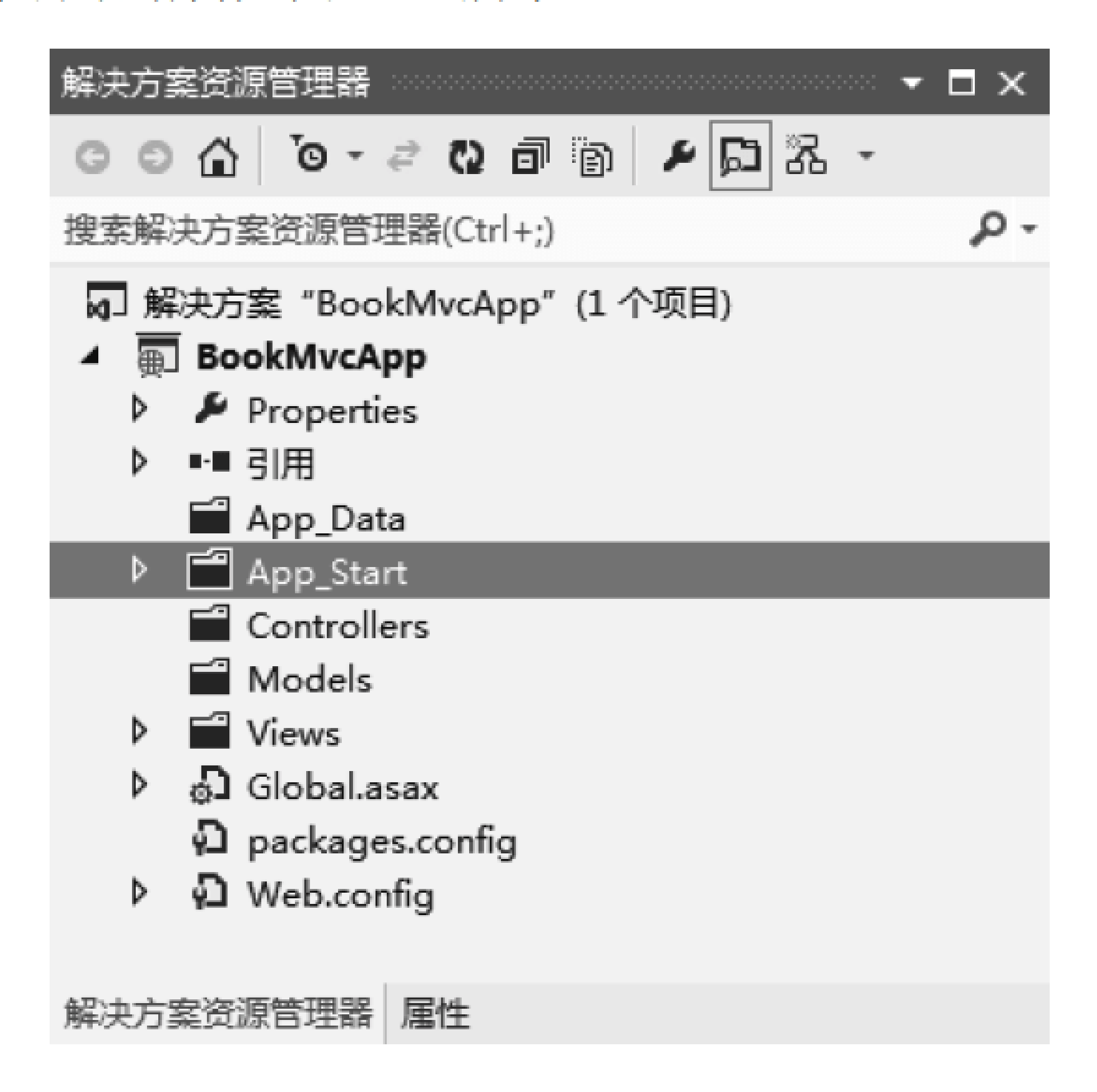


图 15-9　应用程序目录结构

（2）创建一个到图书信息数据库 db_books 的连接，其中的 Books 表保存了图书信息，数据如图 15-10 所示，Id 列是表的主键。



Id	bkno	name	pub	price	tid
2	1002	XML编程技术大全	电子工业出版社	78.5	8
3	1003	Oracle编程入门经典	电子工业出版社	42.8	2
7	1007	VFP实用教程	人民邮电出版社	46.6	2
8	1008	Java测试	人民邮电	39	1
NULL	NULL	NULL	NULL	NULL	NULL

图 15-10　查看 Books 数据表

（3）右击 Models 目录，选择【添加】|【ADO.NET 数据实体模型】命令，在弹出的对话框中将项名称设置为“Book”，再单击【确定】按钮。

（4）在【实体数据模型向导】对话框中选择【从数据库生成】模型，如图 15-11 所示。



图 15-11　选择模型

（5）单击【下一步】按钮，在进入的界面中创建一个到 db_books 数据库的连接，并将实体连接设置为 db_booksEntities，如图 15-12 所示。



图 15-12　创建到 db_books 数据库连接

（6）单击【下一步】按钮，选择模型中要包含的表、视图、存储过程和函数，这里只选择 Books 表，将模型命令空间设置为 booksModel，如图 15-13 所示。



图 15-13　设置模型内容和命名空间

（7）单击【完成】按钮开始生成过程。在生成时会弹出【安全警告】对话框，直接单击【确定】按钮即可，如图 15-14 所示为生成后的 Books 模型。

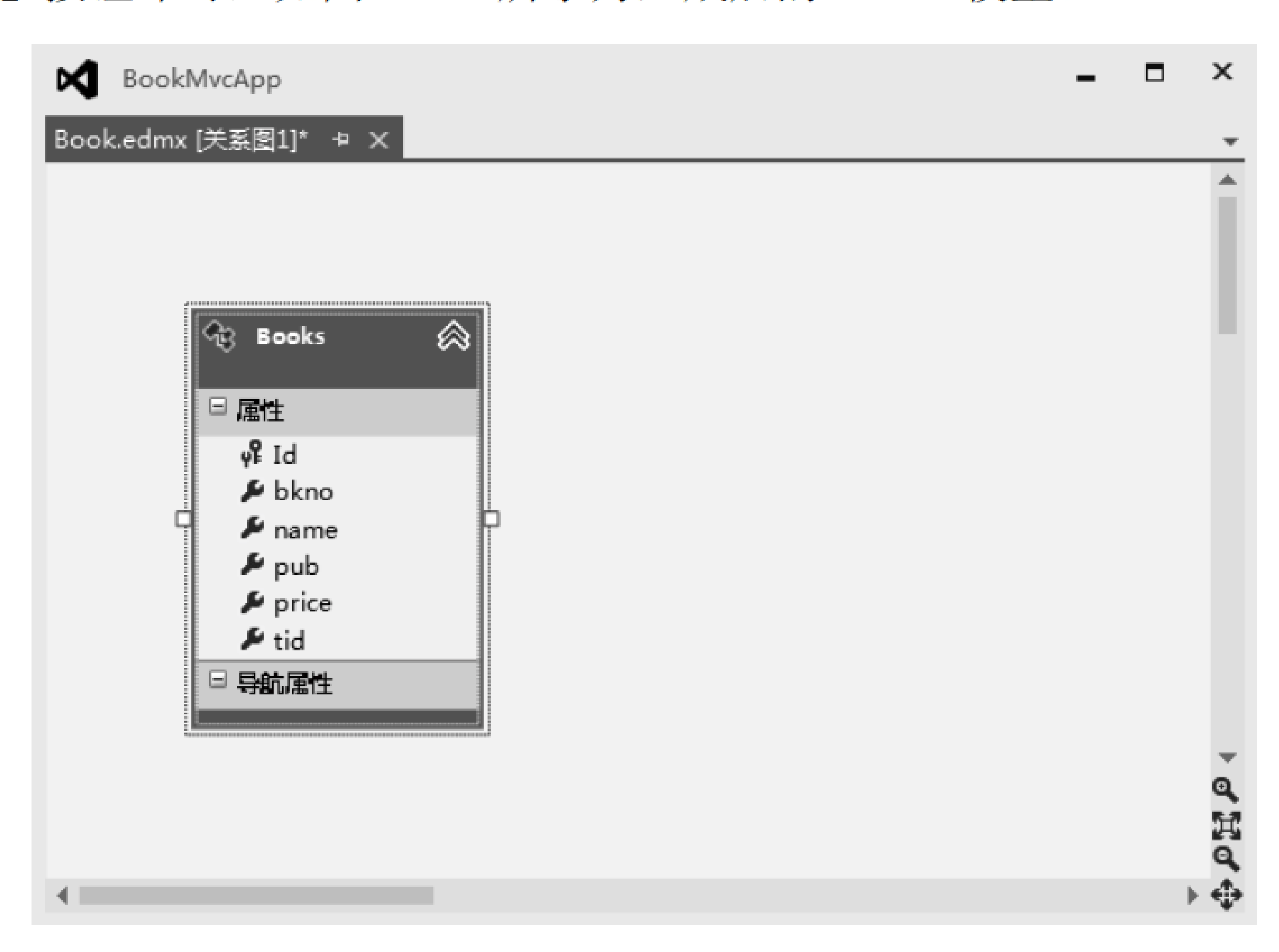


图 15-14　查看 Books 模型

（8）右击 Controllers 目录，选择【添加】|【控制器】命令，打开【添加控制器】对话框，设置控制器名称为 BookController。

（9）在这里为控制器提供了几种不种的控制器模板，可以帮助用户提升开发速度。下面简单了解一下这些模板。

① 空 MVC 控制器。

这是默认的模板，也是最简单的模板。该模板没有提供任何可定制的选项，仅会生成一个包含控制器名称和一个 Index()方法的控制器类。

② 包含读/写操作和视图的 MVC 控制器。

该模板的功能非常丰富，可以针对模板生成访问对象的代码，同时生成了创建、编辑、查看详细和删除视图。在本案例中也是选择该模板。

③ 包含空的读/写操作的 MVC 控制器。

该模板与空 MVC 控制器模板生成的代码一样，不同的是增加了一些针对数据的操作方法，例如 Details、Create、Edit 和 Delete。

④ 空 API 控制器、包含读/操作和视图的 API 控制器和包含空读/写操作的 API 控制器。

这三个模板是与 Web API 相关的内容，与上面介绍的三个模板对应，这里不再详述。

（10）展开【模型类】下拉列表，从中选择生成的 Books 模型，数据上下文类为 db_bookEntities，如图 15-15 所示。



图 15-15 【添加控制器】对话框

（11）设置完成后，单击【添加】按钮开始生成控制器代码。生成过程将在 Controllers 目录下新建 BookController.cs 文件；在 Views 目录下新建 Book 子目录，并在 Books 下新建 Create.cshtml、Delete.cshtml、Details.cshtml、Edit.cshtml 和 index.cshtml 文件。如图

15-16 所示为此时的应用程序目录结构。



图 15-16 生成控制器后的目录结构

（12）现在运行应用程序，将看不到正确结果。这是因为默认生成的 MVC 4 应用程序在运行会请求 Home 控制器的 Index()方法，而该控制器根本不存在。修改方法是打开 App_Start\RouteConfig.cs 文件，将 Home 替换为 Book。

（13）现在运行应用程序将在页面看到 Books 表的内容，如图 15-17 所示。这里因为现在调用的是 Book 控制器的 Index()方法，并使用 Index.cshmtl 文件来呈现页面。



图 15-17 查看 Books 表数据

（14）在生成的页面中同时提供了新增、编辑、查看详细和删除链接。如图 15-18 所示为单击 Edit 链接的效果，如图 15-19 所示为单击 Details 链接的效果。

localhost:20139/Book/Edit/2
Edit
Books
bkno
1002
name
XML编程技术大全
pub
电子工业出版社
price
78.5
tid
8
Save
Back to List

图 15-18　编辑图书信息



图 15-19　查看图书信息

（15）接下来创建一个模版页对生成的视图文件进行美化。首先在应用程序根目录中新建一个 Content 目录用于存放模板页用到的 CSS 和图片文件。

（16）右击应用程序\Views 目录，选择【添加】|【视图】命令，打开【添加视图】对话框，创建一个名称为_ViewStart，使用 Razor 视图引擎的文件，如图 15-20 所示。



图 15-20 【添加视图】对话框

（17）在 Views 目录下新建 Shared 子目录，并在 Shared 目录中添加视图文件_Layout.cshtml。

（18）打开_ViewStart.cshtml 文件，使用 Razor 指令指定模板页的文件，代码如下。

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

（19）打开_Layout.cshtml 文件，使用标准 HTML 代码进行布局。其中关键代码如下。

```
<h2><img src="/Content/imgs/bullet1.gif" />@ViewBag.Message</h2>
 @RenderBody()
```

@ViewBag.Message 表示调用 Message 变量，@RenderBody()是一个表示视图内容的占位符。

（20）打开 Book 控制器的 Index()方法，在这里为 Message 变量指定一个字符串，代码如下。

```
public ActionResult Index()
{
    ViewBag.Message = "查看所有图书信息";          //设置一个标题
    return View(db.Books.ToList());                //获取图书信息列表
}
```

（21）打开 Index()方法对应的视图文件 Index.cshtml。对页面的呈现效果进行修改，最终代码如下。

```
@model IEnumerable<BookMvcApp.Models.Books>
@{
    ViewBag.Title = "Index";
}
<table border="1" >
    <tr>
        <th>编号</th>
        <th>图书名称</th>
        <th> 出版社</th>
        <th>价格</th>
        <th> 分类编号</th>
        <th>操作</th>
    </tr>
@foreach (var item in Model) {
    <tr>
        <td> @Html.DisplayFor(modelItem => item.bkno) </td>
        <td> @Html.DisplayFor(modelItem => item.name) </td>
        <td> @Html.DisplayFor(modelItem => item.pub) </td>
        <td> @Html.DisplayFor(modelItem => item.price) </td>
        <td> @Html.DisplayFor(modelItem => item.tid) </td>
        <td>
            @Html.ActionLink("编辑", "Edit", new { id=item.Id }) |
            @Html.ActionLink("详情", "Details", new { id=item.Id }) |
            @Html.ActionLink("删除", "Delete", new { id=item.Id })
        </td>
    </tr>
}
</table>
<hr>  @Html.ActionLink("新增图书信息", "Create")
```

（22）再次运行应用程序将看到美化后的图书信息列表，如图 15-21 所示。

（23）使用相同的方法对 Create.cshtml、Delete.cshtml、Details.cshtml 和 Edit.cshtml 文件进行布局的美化。



图 15-21 美化后的图书信息列表

（24）从首页中单击【新建图书信息】链接请求 Book 控制器的 Create 视图，页面效果如图 15-22 所示。



图 15-22 添加一本图书信息

（25）从首页中单击【删除】链接请求 Book 控制器的 Delete 视图，页面效果如图 15-23 所示。至此，使用 ASP.NET MVC 4 框架实现对图书信息的管理就完成了。



图 15-23 删除图书信息

思考与练习

一、填空题

1. MVC 设计模式将应用程序按用户界面的功能划分为模型、视图和__________三个模块。

2. 在创建一个 ASP.NET MVC 应用程序后，__________目录用于存储控制器文件。

3. ASP.NET MVC Web 应用程序使用__________文件中的代码来设置全局配置信息。

4. 在 Razor 中要定义一个代码块需要以__________开始，以}结束。

5. <%: Html.LabelFor(m => m.UserName) %>转换成 Razor 的等效代码是__________。

6. ASP.NET MVC Web 应用程序使用__________文件中的代码来设置全局 URL 路由默认值。

二、选择题

1. ASP.NET MVC 是一个__________。
   A. 设计思想
   B. 类
   C. 框架
   D. 设计模式

2. 下列说法错误的是__________。
   A. ASP.NET MVC 中 View 默认放在 Views 目录下面，也可以是其他目录
   B. ASP.NET MVC 中 Model 必须放在 Models 目录下面
   C. ASP.NET MVC 中 View 必须放在 Views 目录下面
   D. ASP.NET MVC 中 Controller 默认必须放在 Controllers 目录下面

3. 默认的 ASP.NET MVC 4 应用程序访问路径是__________。
   A. /Home/Index
   B. Default.aspx
   C. Home.aspx
   D. Index

4. 如下哪个 MVC 应用程序模板适用于与 Ajax 的交互？__________
   A. 移动应用程序模板
   B. Web API 模板
   C. 基本模板

D．空模板

5．下面关于 Razor 视图引擎的描述，不正确的是__________。

A．Razor 是一门新的编程语言

B．Razor 支持所有文本编辑器

C．Razor 的作用是渲染 HTML

D．Razor 的语法更加易于输入、阅读和维护，且结构更简洁

6．__________目录用于保存应用程序用到的配置文件后台代码。

A．App_Start

B．App_Codes

C．Controllers

D．Config

三、简答题

1．简述什么是 MVC，它有什么优缺点。

2．MVC 与 ASP.NET MVC 之间有什么关系？

3．简述 Razor 视图引擎。

4．简述创建一个 MVC 应用程序的步骤。

5．罗列 MVC 应用程序的核心目录及其作用。

6．如何理解 MVC 应用程序的运行流程？

第 16 章　WCF 入门

WCF 是一个基于 SOA（Service Oriented Architecture，面向服务架构）的.NET 平台下的统一框架，它代表了软件架构与开发的一种发展方向。

WCF 使开发人员可以用最少的时间建立软件与外界通信的模型。它整合了.NET 平台下所有和分布式系统有关的技术，如 Web Service、.NET Remoting、WSE 和 MSMQ 等。这使得开发者能够建立一个跨平台的、安全的、可依赖的、事务性的解决方案，并且能与已有系统兼容协作。

通过本章的学习，读者将了解 WCF 的概念、WCF 核心组成部分以及 WCF 的创建和调用，最后介绍了防盗链的实现。

本章学习要点：

- 了解 WCF 的作用和组成部分
- 掌握创建 WCF 项目的方法
- 了解 WCF 的测试方法
- 理解 WCF 中地址、绑定和合约的作用和设置方法
- 熟悉使用端点来配置 WCF 服务
- 熟悉防盗链的实现

16.1　WCF 概述

WCF 是 Windows Communication Foundation 的缩写，中文含义为 Windows 通信基础。WCF 最开始由.NET Framework 3.0 引入，与 WPF 和 WF 组成了三大开发类库。WCF 的最终目标是通过进程或者不同的系统、通过本地网络或者通过 Internet 收发客户和服务之间的消息。

16.1.1　WCF 简介

WCF 是一个面向服务编程的分布式架构。它是.NET 框架的一部分，因为 WCF 并不能脱离.NET 框架而单独存在。因此，虽然 WCF 是微软为应对 SOA 解决方案的开发需要而专门推出的，但它并不是像 Spring 和 Struts 那样的框架，也不是像 EJB 那样的容器或者服务器。

WCF 是微软对分布式编程技术的最大集成者，它将 DCOM、.NET Remoting、Enterprise Service、Web Service、WSE 以及 MSMQ 集成在一起，从而降低了学习分布式系统开发者的难度曲线，并统一了开发标准。

也就是说，在 WCF 框架下开发基于 SOA 的分布式系统变得容易了。微软将所有与

此相关的技术要素都包含在内，掌握了 WCF 就相当于掌握了敲开 SOA 大门的钥匙。

例如，下面通过一个实例说明 WCF 的优势所在。假设要为一家汽车租赁公司开发一个新的应用程序，用于租车预约服务。该租车预约服务会被多种应用程序访问，包括呼叫中心，基于 J2EE 的租车预约服务以及合作伙伴的应用程序，如图 16-1 所示。

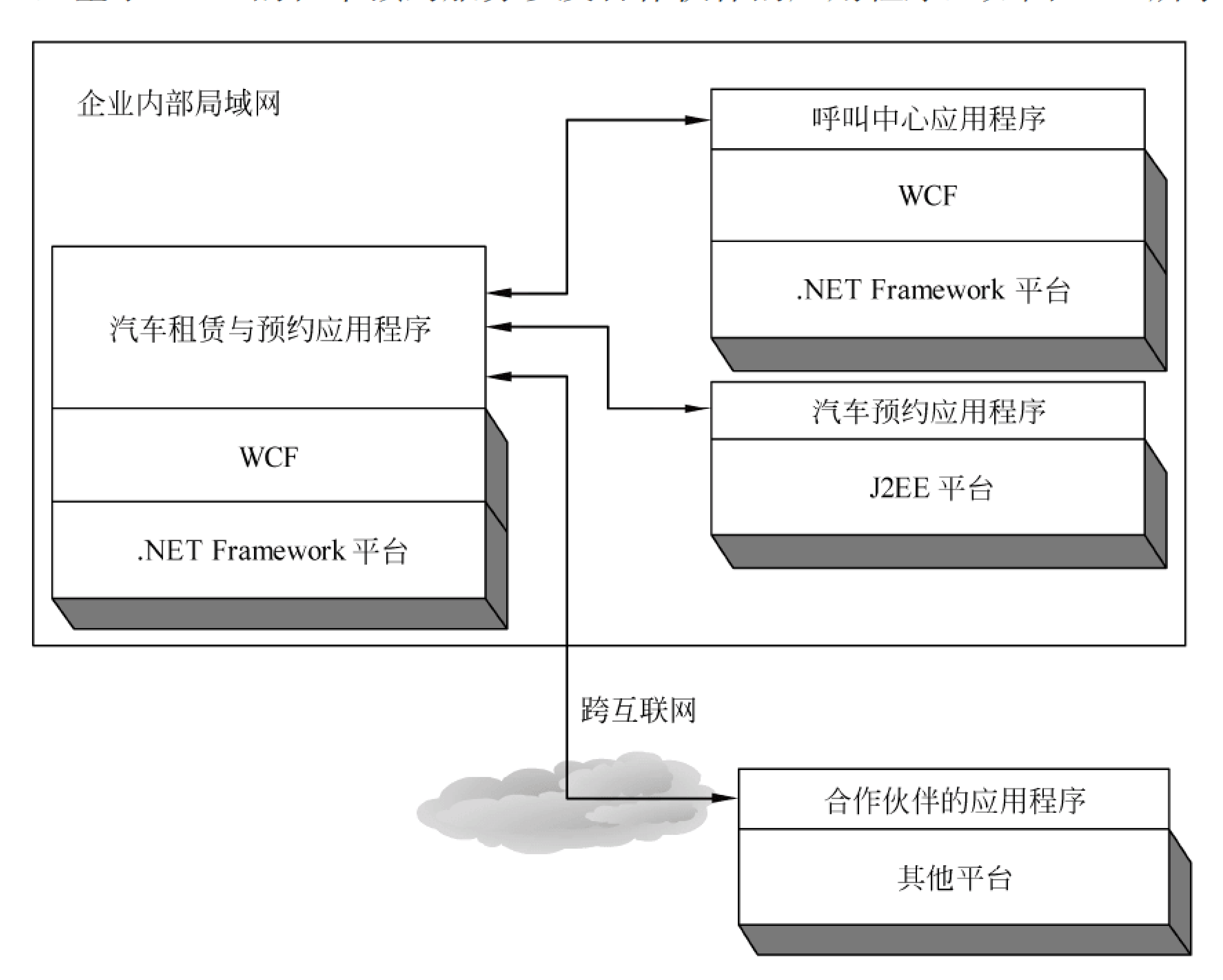


图 16-1 汽车租赁公司应用程序分布图

呼叫中心运行在 Windows 平台下，是在.NET Framework 下开发的应用程序，用户为公司员工。由于该汽车租赁公司兼并了另外一家租赁公司，该公司原有的汽车预约服务应用程序是 J2EE 应用程序，运行在非 Windows 操作系统下。呼叫中心和已有的汽车预约应用程序都运行在企业内部局域网环境下。合作伙伴的应用程序可能会运行在各种平台下，这些合作伙伴包括婚纱拍摄公司、司仪策划公司等，他们会通过 Internet 来访问汽车预约服务，实现对汽车的租用。

这样一个案例是一个典型的分布式应用系统。如果没有 WCF，利用.NET 现有的技术应该如何开发呢？

首先考虑呼叫中心，它和要开发的汽车预约服务一样，都是基于.NET Framework 的应用程序。呼叫中心对于系统的性能要求较高，在这样的前提下，.NET Remoting 是最佳的实现技术。它能够高性能地实现.NET 与.NET 之间的通信。

要实现与已有的 J2EE 汽车预约应用程序之间的通信，只有基于 SOAP 的 Web Service 可以实现此种目的，它保证了跨平台的通信；而合作伙伴由于是通过 Internet 来访问，利用 ASP.NET Web Service，也是较为合理的选择，它保证了跨网络的通信。由于涉及网络之间的通信，还要充分考虑通信的安全性，利用 WSE（Web Service Enhancements）可以为 Web Service 提供安全的保证。

一个好的系统除了要保证访问和管理的安全和高性能，同时还要保证系统的可依赖性。因此，事务处理是企业应用必须考虑的因素，对于汽车预约服务，同样如此。在.NET中，Enterprise Service（COM+）提供了对事务的支持，其中还包括分布式事务（Distributed Transactions）。不过对于 Enterprise Service 而言，它仅支持有限的几种通信协议。

如果还要考虑异步调用、脱机连接、断点连接等功能，还需要应用 MSMQ（Microsoft Message Queuing）利用消息队列支持应用程序之间的消息传递。

如此看来，要建立一个好的汽车租赁预约服务系统，需要用到的.NET 分布式技术包括：.NET Remoting、Web Service、COM+等 5 种技术，这既不利于开发者的开发，也加大了程序的维护难度和开发成本。正是基于这样的缺陷，WCF 才会在.NET 3.0 中作为全新的分布式开发技术被微软强势推出，它整合了上述介绍的分布式技术，成为理想的分布式开发解决方案。如表 16-1 所示列出了 WCF 与之前相关技术的比较。

表 16-1 WCF 与现有技术比较

特性	Web Service	.NET Remoting	Enterprise Services	WSE	MSMQ	WCF
具有互操作性的 Web 服务	支持					支持
支持.NET 之间的通信		支持				支持
分布式事务			支持			支持
支持 WS 标准				支持		支持
消息队列					支持	支持

从表 16-1 中来看，WCF 完全可以看作是 Web Service、.NET Remoting、Enterprise Service、WSE、MSMQ 等技术的并集。因此，对于上述汽车预约服务系统的例子，利用 WCF 就可以解决包括安全、可依赖、互操作、跨平台通信等需求。开发者再不用去分别了解.NET Remoting 和 WSE 等各种技术了。

注 意

事实上 WCF 远非简单的并集这样简单，它是真正面向服务的产品，它已经改变了通常的开发模式。

16.1.2 WCF 组成部分

软件设计的一个重要原则：软件组件必须针对特定的任务专门设计和优化。假如要做一个管理软件，想象一下，如果一个软件非常依赖于与外界通信，那么不能把管理软件与外界通信的逻辑考虑在管理系统内部。所以，必须把通信任务委托给不同的组件。用 WCF 术语来说，这个组件称为 WCF 服务。更通俗地讲，WFC 服务就是负责与外界通信的软件。

一个 WCF 服务由下面三部分组成。

（1）Service Class：一个标记了 ServiceContract 属性的类，其中可以包含多个方法。该类除了标记一些 WCF 特有的属性外，与一般的类没有区别。

（2）Host：可以是应用程序、进程或者 Windows 服务等，它是 WCF 服务的运行

环境。

（3）Endpoints：可以是一个，也可以是一组，它是 WCF 实现通信的核心要素。

如图 16-2 所示显示了这三部分在 WCF 服务中的位置。

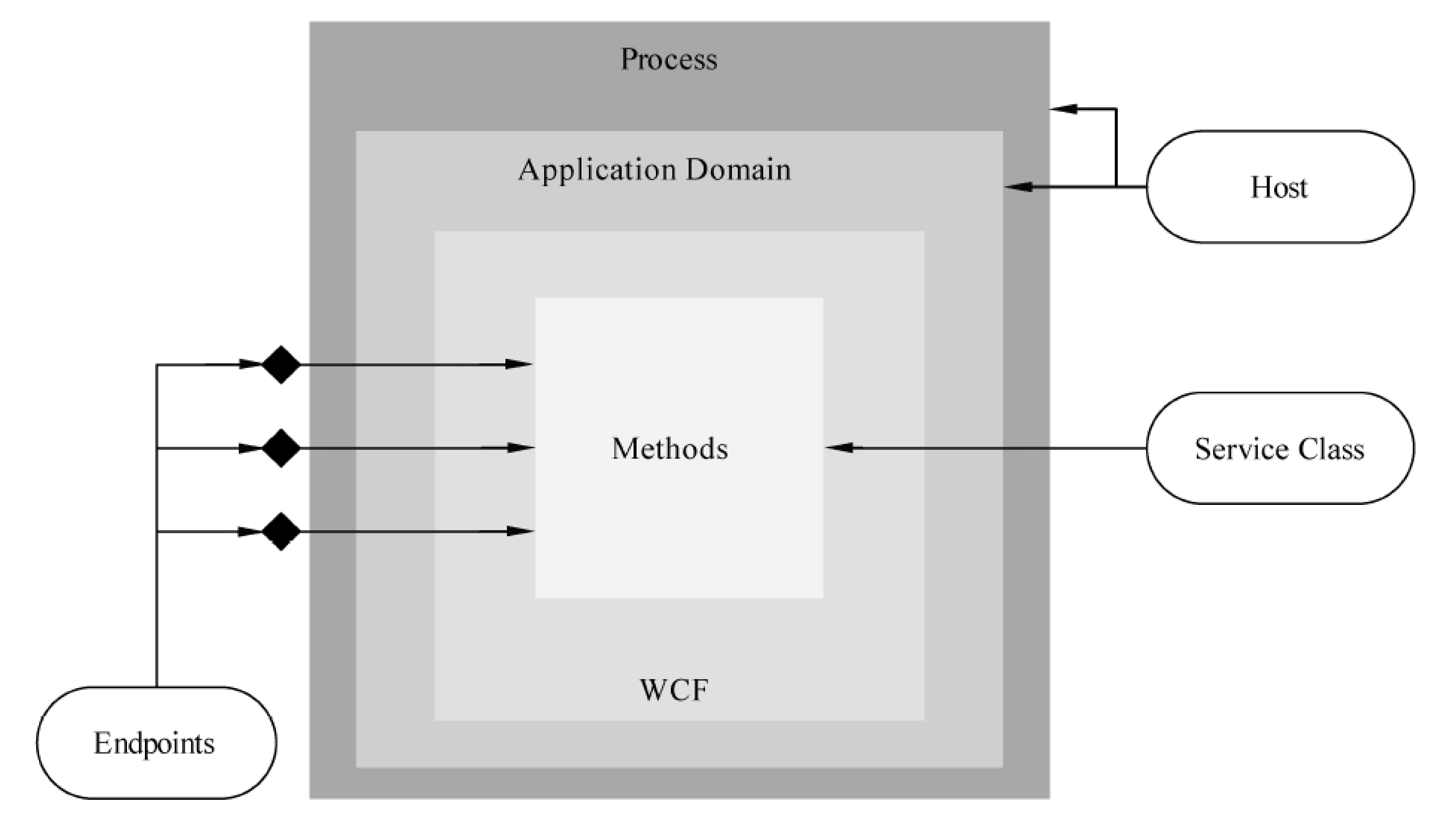


图 16-2 WCF 组成部分

一个 WCF 服务必须能为不同的通信场景提供不同的访问点，这些访问点称为 WCF 端点，也就是上面所提到的 EndPoint。每个端点都有一个绑定（Binding）、一个地址（Address）和一个合约（Contract），如图 16-3 所示。

（1）绑定：指定该端点如何与外界通信，也就是为端点指定通信协议，包括传输协议、编码协议和安全协议。

（2）地址：一个端点地址，如果通过端点与 WCF 通信，必须把通信指定到网络地址。

（3）合约：一个端点上合约指定通过该端点的用户能访问到 WCF 服务的什么操作。

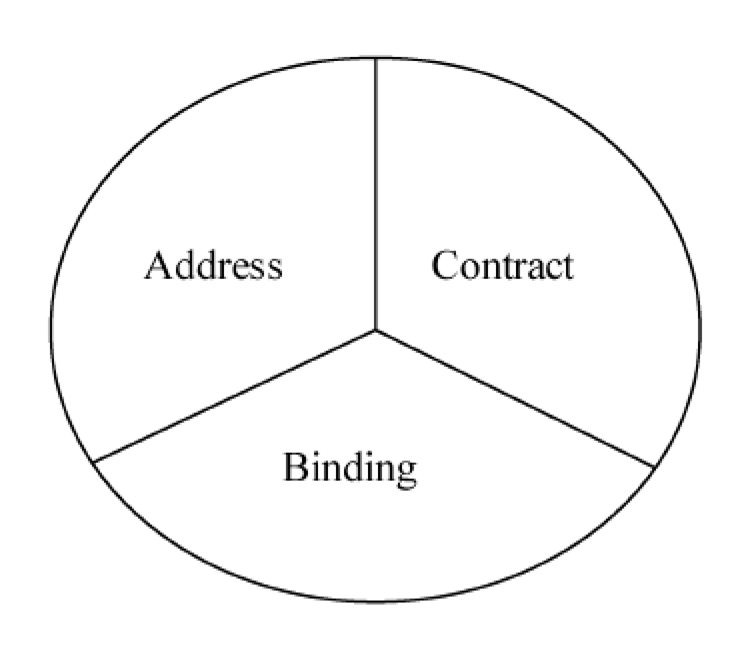


图 16-3 端点组成部分



为了便于记忆 EndPoint 的这三个部分，可以将 Address、Binding 和 Contract 简称为“ABC”。

16.2 实践案例——创建第一个 WCF 服务程序

通过 16.1 节的学习，相信读者对 WCF 的概念、作用及其组成部分都有了一个大致的了解。下面通过一个简单的 WCF 案例，帮助读者了解 WCF 程序的开发步骤和运行流程，为后面核心元素的理解奠定基础。

（1）在 VS 2012 中新建一个空白解决方案，名称为 wcfDemo。

（2）向解决方案中添加一个类型为【WCF 服务应用程序】的项目，定义名称为 WcfService1，单击【确定】按钮完成创建，如图 16-4 所示。

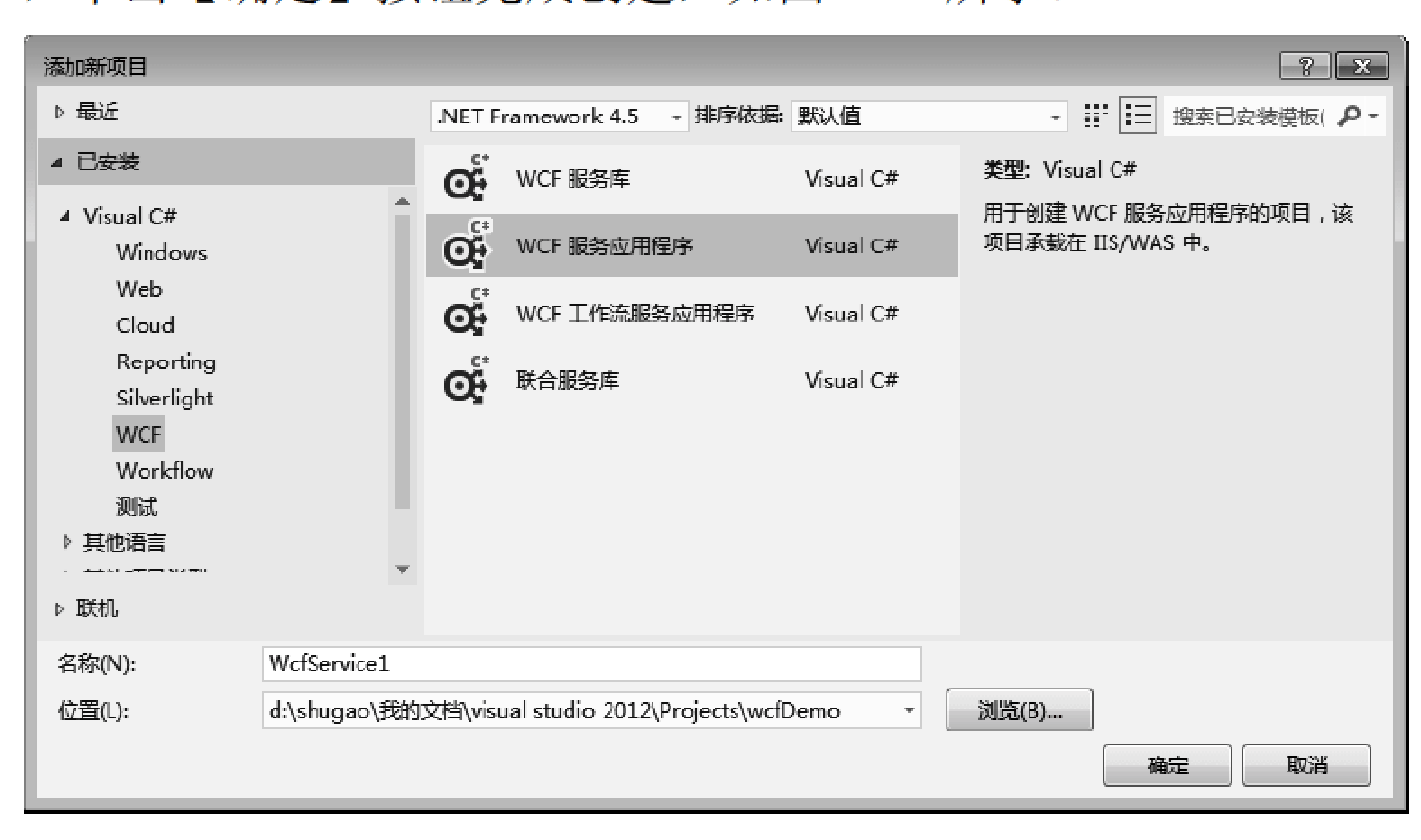


图 16-4 创建 WCF 项目

（3）WCF 项目创建完成之后，在【解决方案资源管理器】窗口中将看到默认生成的项目结构及 IService1.cs 文件的内容，如图 16-5 所示。

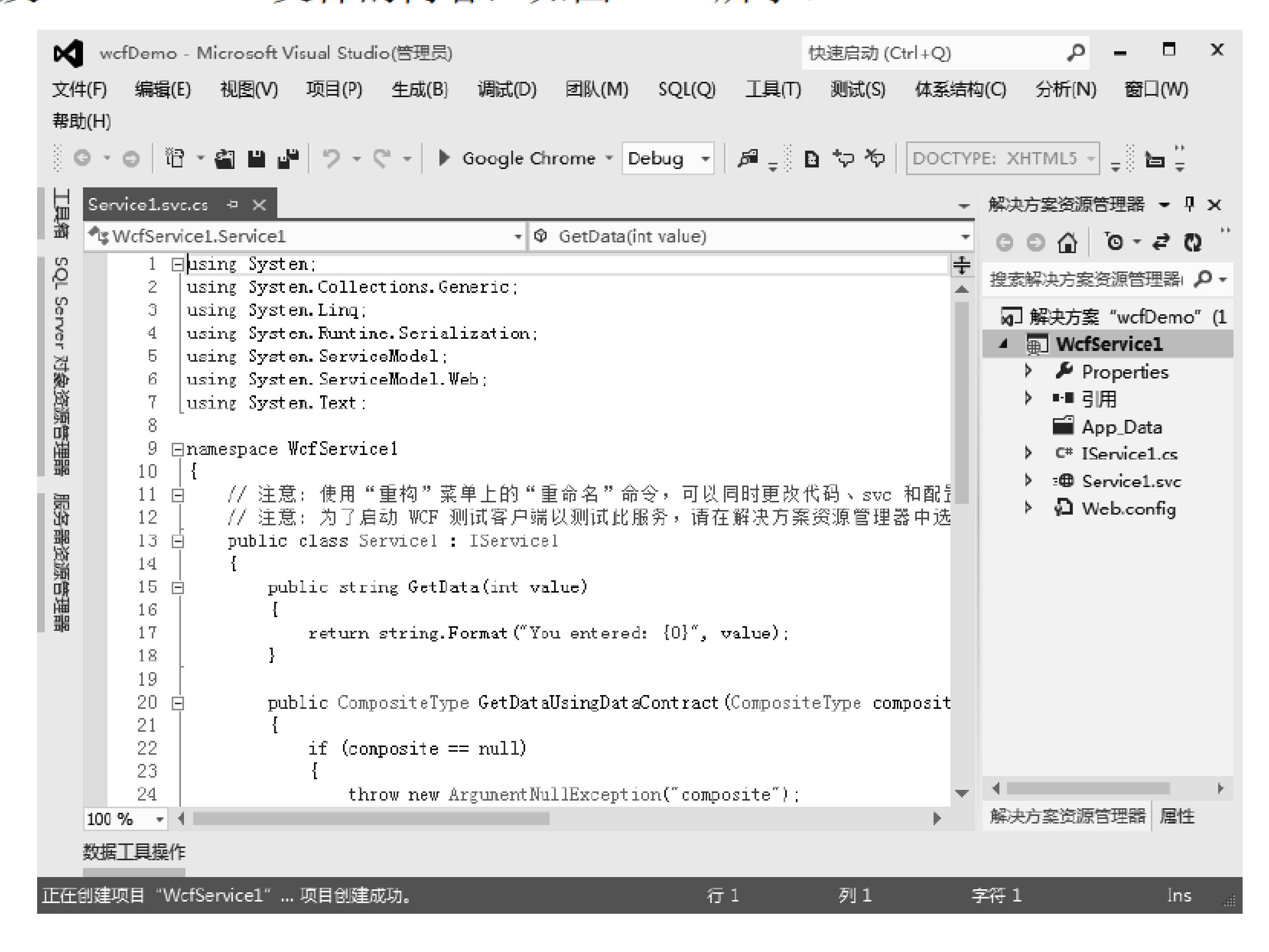


图 16-5 WCF 项目内容

这里的 IService1.cs 称为合约，WCF 的合约必须要以接口的方式定义。Service1.svc 为服务的实现，即它继承接口并进行实现。Web.config 是 WCF 服务的配置信息。

（4）在 IService1.cs 文件中的“[ServiceContract]”为服务合约，“[OperationContract]”声明该方法可以在 WCF 中调用。按照这种规则，手动创建一个方法，代码如下。

```
//手动创建的方法
[OperationContract]
string Welcome(string str);
```

（5）打开 Service1.svc 文件，添加服务合约中声明的 Welcome()方法的实现代码。具体代码如下所示。

```
//实现在服务合约中定义声明的 Welcome()方法
public string Welcome(string str)
{
    return string.Format("你好 {0}，欢迎来到 WCF 的世界。当前时间是：{1}。", str,
    DateTime.Now.ToString());
}
```

（6）至此，关于 WCF 服务合约的创建及实现就都定义好了。下面对 WCF 服务的配置信息进行定义，指定客户端通过什么方式引用 WCF 服务。打开 Web.config 文件，其中的内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext" value=
    "true" />
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5"/>
  </system.web>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- 为避免泄漏元数据信息，请在部署前将以下值设置为 false -->
          <serviceMetadata httpGetEnabled="true" httpsGetEnabled=
          "true"/>
          <!-- 要接收故障异常详细信息以进行调试，请将以下值设置为 true。在部署前设
          置为 false 以避免泄漏异常信息 -->
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>
```

```
      </behaviors>
      <protocolMapping>
         <add binding="basicHttpsBinding" scheme="https" />
      </protocolMapping>
      <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
      multipleSiteBindingsEnabled="true" />
    </system.serviceModel>
    <system.webServer>
      <modules runAllManagedModulesForAllRequests="true"/>
      <!--
         若要在调试过程中浏览 Web 应用程序根目录，请将下面的值设置为 True。
         在部署之前将该值设置为 False 可避免泄漏 Web 应用程序文件夹信息。
        -->
      <directoryBrowse enabled="true"/>
    </system.webServer>
  </configuration>
```

（7）现在运行上面创建的 WCF 项目 WcfService1。VS 2012 检测到当前要运行的是 WCF 项目会打开【WCF 测试客户端】工具。在这里可以浏览 WCF 服务的地址、提供的方法、配置文件，而且还可以对方法进行测试。如图 16-6 所示为测试 Welcome()方法时的窗口。



图 16-6 使用【WCF 测试客户端】测试 Welcome()方法

（8）与 Web 服务类似，WCF 服务也可以在浏览器中浏览。例如，在本实例中可以通过地址 http://localhost:22872/Service1.svc 来浏览 WCF 服务，如图 16-7 所示。



图 16-7 使用浏览器查看 **WCF** 服务

如果输入地址“http://localhost:22872/Service1.svc?singleWsdl”将可以看到 WCF 服务的 WSDL 内容，如图 16-8 所示。

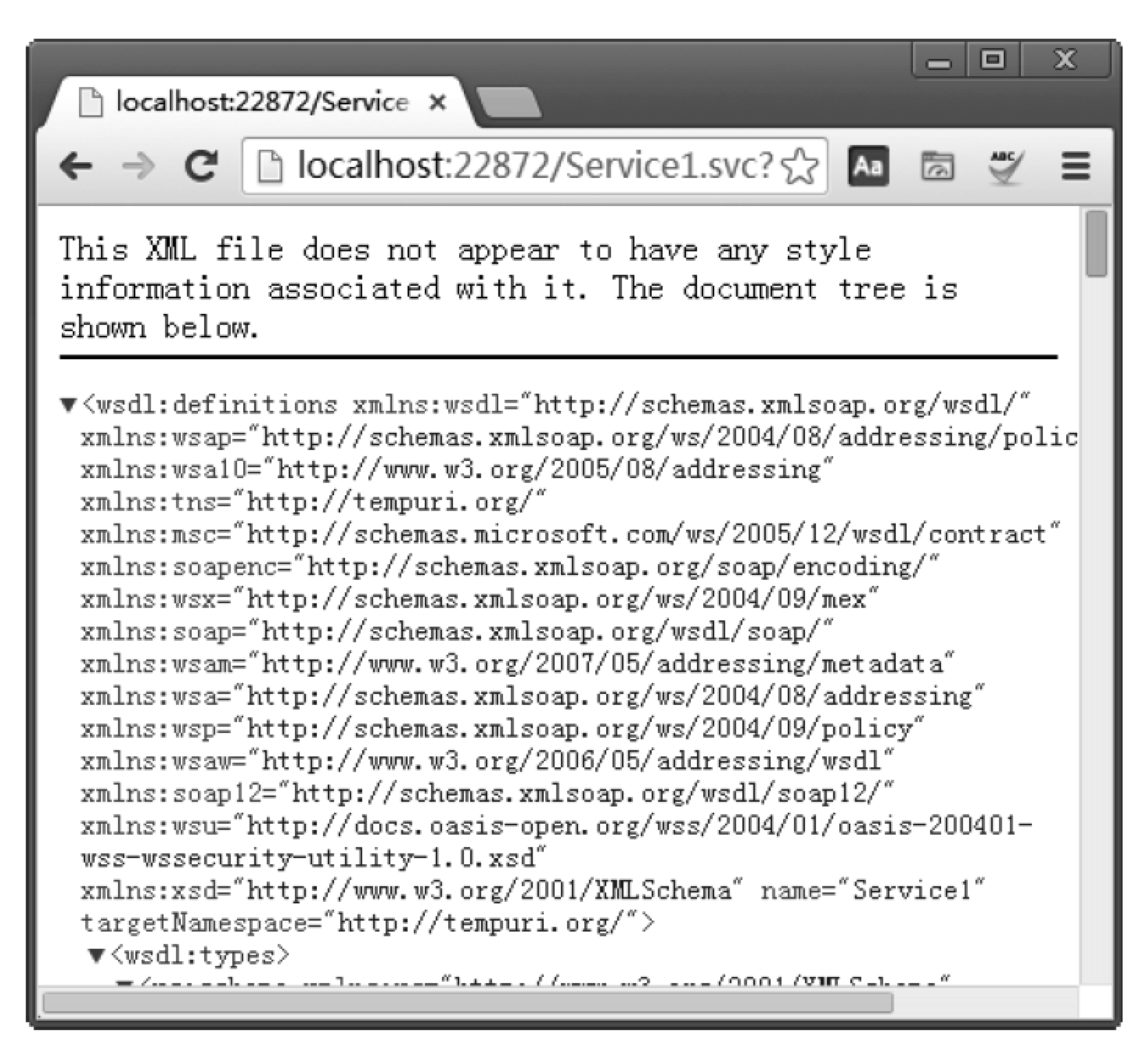


图 16-8 查看 **WCF** 服务的 **WSDL**

（9）向解决方案中添加一个名为 testWcfWeb 的 Web 应用程序项目，使用此项目调用 WCF 服务来进行测试。在【解决方案资源管理器】窗格中右击 testWcfWeb 项目，选

择【添加服务引用】命令，打开【添加服务引用】对话框。

（10）单击【发现】按钮，选择【解决方案中的服务】项。然后从【服务】列表中选择 Service1.svc 文件中的 Service1 类 IService1，此时右侧的【操作】列表会显示所有可用的方法，并可以定义命名空间，这里使用默认值，如图 16-9 所示。

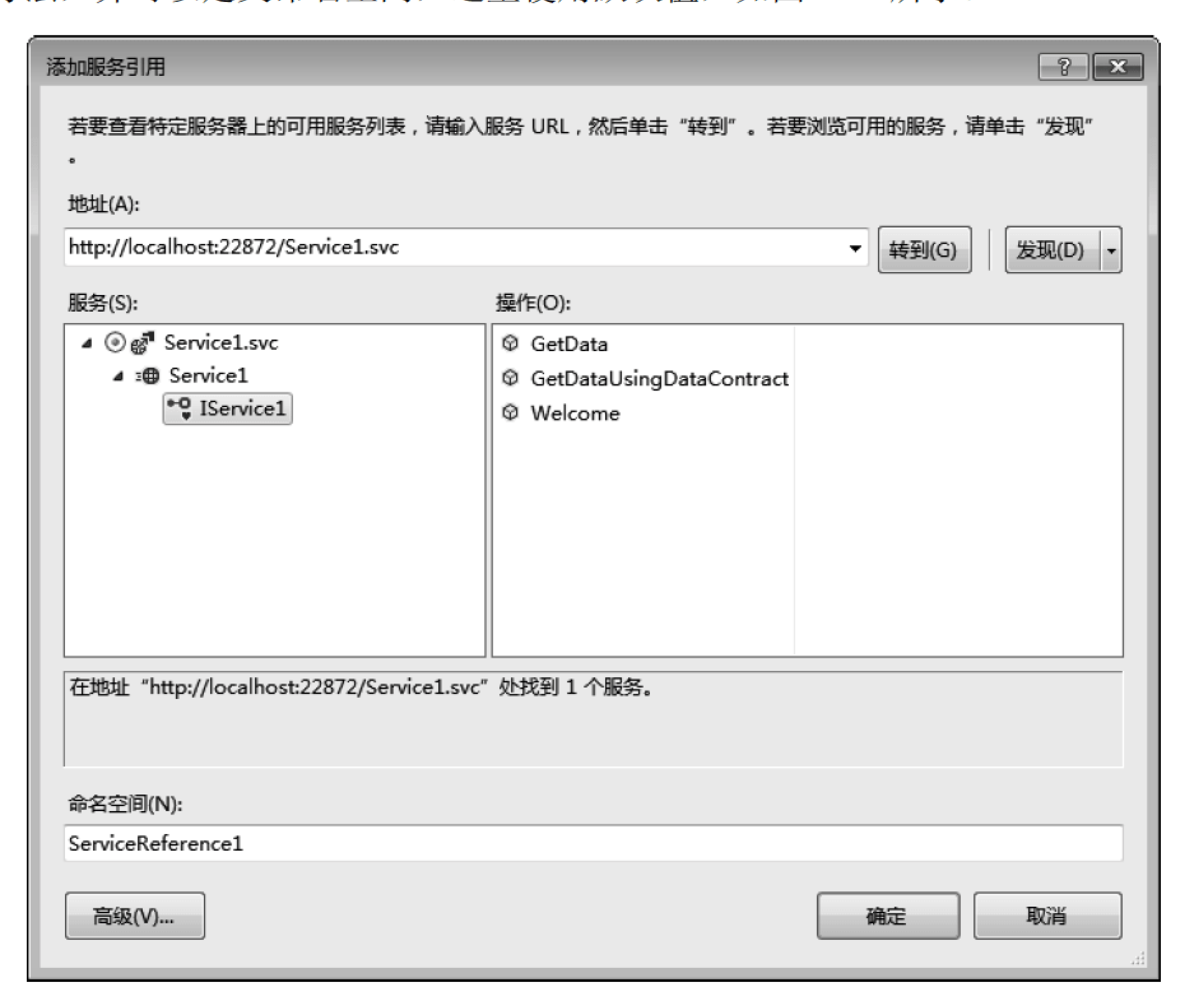


图 16-9 【添加服务引用】对话框

（11）打开 testWcfWeb 项目的 Web.config 文件，会发现生成如下与 WCF 服务有关的代码。

```
<system.serviceModel>
    <bindings>
        <basicHttpBinding>
            <binding name="BasicHttpBinding IService1" />
        </basicHttpBinding>
    </bindings>
    <client>
        <endpoint address="http://localhost:22872/Service1.svc" binding=
        "basicHttpBinding"
            bindingConfiguration="BasicHttpBinding_IService1"
            contract="ServiceReference1.IService1" name=
            "BasicHttpBinding IService1" />
    </client>
</system.serviceModel>
```

上述代码中的 http://localhost:22872/Service1.svc 为 WCF 服务的引用地址。

（12）在 Web 应用程序中添加一个 WebForm1.aspx 文件，并添加如下代码。

```
<h1>测试 WCF 服务</h1>
<p> 请输入您的名称:
    <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
    <asp:Button ID="Button1" runat="server" Text="确定" OnClick=
    "Button1 Click" />
    <asp:Literal ID="ltTarget" runat="server"></asp:Literal>
</p>
```

（13）为【确定】按钮创建 Click 事件，实现调用 WCF 服务中的 Welcome()方法并显示结果。代码如下。

```
protected void Button1 Click(object sender, EventArgs e)
 {
    //创建对 WCF 服务库的引用对象 sc
    ServiceReference1.Service1Client sc = new ServiceReference1.
    Service1Client();
    //调用 WCF 服务库中的 Welcome()方法，将结果保存到 result
    string result=sc.Welcome(txtName.Text);
    ltTarget.Text = result;            //显示结果
 }
```

（14）在浏览器中可看 WebForm1.aspx 页面，运行效果如图 16-10 所示。



图 16-10　WebForm1.aspx 页面调用 WCF 效果

16.3 WCF 核心元素

WCF 框架中包含大量的基础概念，在 16.2 节中创建了一个简单的 WCF 程序，本节将对前面出现的 WCF 核心概念进行详细介绍，如 WCF 地址、绑定和合约等。

16.3.1 地址

WCF 的每一个服务都具有一个唯一的地址（Address）。每个地址都包含两个重要元素：服务位置与传输协议，或者是用于服务通信的传输方式。服务位置包括目标主机名、站点（或者网络）、通信端口、管道（或者队列），以及一个可选的特定路径或者 URI。

总地来说，WCF 支持如下几种传输方式：HTTP，TCP，Peer Network（对等网），IPC（基于命名管道的内部进程通信），MSMQ。

地址通常采用如下格式。

```
[传输协议]://[主机名或域名][:可选端口]
```

例如，如下所示的这些都是正确的地址。

```
http://localhost:8080
http://localhost:5678/MyWcfService
net.tcp://localhost:1010/MyWcfService
net.pipe://localhost/MyWcfPipe
net.msmq://localhost/public/MyWcfService
net.msmq://localhost/MyWcfService
```


如上述示例所示，可以将地址"http://localhost:8080"理解为"采用 HTTP 访问 localhost 主机，并在 8080 端口等待用户的调用"。对于"http://localhost:5678/MyWcfService"地址则可以理解为"采用 HTTP 访问 localhost 主机，MyWcfService 服务在 5678 端口处等待用户的调用。"

1. TCP 地址

TCP 地址采用 net.tcp 作为协议进行传输，通常它还带有端口号。例如：

```
net.tcp://localhost:8018/MyWcfService
```

如果没有指定端口号，则 TCP 地址的默认端口号为 808。例如：

```
net.tcp://localhost/MyWcfService
```

另外，两个 TCP 地址（来自于相同的主机）可以共享一个端口。例如：

```
net.tcp://localhost:8018/MyWcfService
net.tcp://localhost:8018/MyOtherWcfService
```

2. HTTP 地址

HTTP 地址是使用频率最高的一种地址，它使用 HTTP 进行传输，也可以利用 HTTPS 进行安全传输。HTTP 地址通常会被用作对外的基于 Internet 的服务，并为其指定端口号。例如：

```
http://localhost:8088
```

```
http://localhost:8088/MyWcfService
https://localhost/MyWcfService
```

如果没有指定端口号，则默认为 80。与 TCP 地址相似，两个相同主机的 HTTP 地址可以共享一个端口，甚至相同的计算机。

3. IPC 地址

IPC 地址使用 net.pipe 协议进行传输，这意味着它将使用 Windows 的命名管道机制。在 WCF 中，使用命名管道的服务只能接收来自同一台计算机的调用。

因此，在使用时必须明确指定本地计算机名或者直接命名为 localhost，然后再为管道名提供一个唯一的标识字符串。例如：

```
net.pipe://localhost/MyWcfPipe
net.pipe://zhht/MyWcfPipe
```



每台计算机只能打开一个命名管道。因此，两个命名管道地址在同一台计算机上不能共享一个管道名。

4. MSMQ 地址

MSMQ 地址使用 net.msmq 协议进行传输，即使用了微软消息队列（Microsoft Message Queue）机制。在使用时必须为 MSMQ 地址指定队列名。如果是处理私有队列，则必须指定队列类型。例如：

```
net.msmq://localhost/private/MyWcfService
net.msmq://localhost/private/MyOtherWcfService
```

但对于公有队列而言，队列类型可以省略。例如：

```
net.msmq://localhost/MyWcfService
net.msmq://zhht/private/MyWcfService
```

5. 对等网地址

对等网地址（Peer Network Address）使用 net.p2p 协议进行传输，它使用了 Windows 的对等网传输机制。如果没有使用解析器，就必须为对等网地址指定对等网名称、唯一的路径以及端口。

16.3.2 绑定

WCF 中的绑定（Binding）指定了服务的通信方式。

使用绑定也是 WCF 开发区别于 Web 服务开发的一个重要方面。因为 WCF 带有许多可供选择的绑定，每种绑定都适合于特定的需求。另外，如果现有的绑定类型不能满

足需求，还可以通过扩展 CustomBinding 类型创建绑定。

简单来说，WCF 绑定可以指定如下特性。

（1）传输协议。

（2）安全要求。

（3）编码格式。

（4）事务处理要求。

一个绑定类型包含多个绑定元素，它们描述了上面所有的绑定要求。WCF 内置了 9 种类型的绑定，如表 16-2 所示列出了这些绑定类型及其说明。

表 16-2 WCF 绑定类型

绑定类型	说明
BasicHttpBinding	BasicHttpBinding 在 Web 服务的交互操作中使用最广泛，可使用 HTTP 或者 HTTPS 进行传输，其安全性取决于协议安全
WSHttpBinding	WSHttpBinding 是对 BasicHttpBinding 的增强，它使用扩展的 SOAP 确保安全性、可靠性和事务处理。同样使用 HTTP 或者 HTTPS 进行传输，但是为了确保安全，使用了 WS-Security
WSFederationHttpBinding	WSFederationHttpBinding 是一种安全、可交互操作的绑定，它支持在多个系统上共享身份，以进行身份验证和授权
WSDualHttpBinding	与 WSHttpBinding 相反，这种类型支持消息的双向传输
NetTcpBinding	使用 TCP/IP 为通信提供绑定
NetPeerTcpBinding	使用对等网协议为通信提供绑定
NetNamedPipeBinding	使用命名管道协议为通信提供绑定，该类型为在同一系统的不同进程之间的通信进行了优化
NetMsmqBinding	该类型的绑定会将消息发送到消息队列中，它要求 WCF 应用程序位于客户端和服务器上
MsmqIntegrationBinding	该类型的绑定将会使用消息队列的已有应用程序
CustomBinding	这种类型是自定义的绑定类型

提示

所有以 Net 前缀开始的绑定类型都使用二进制编码在.NET 应用程序之间通信，这种编码格式比文本格式要快。

在表 16-2 中列出的每种绑定类型都有自己的特性。例如，以 WS 为前缀的绑定类型是独立于平台的，支持 Web 服务规范。以 Net 为前缀的使用二进制格式，使.NET 应用程序之间的通信具有更高的性能。如表 16-3 所示针对这些绑定类型按特性进行了分类。

表 16-3 绑定类型支持的特性

特性	支持的绑定类型
会话	WSHttpBinding、WSDualHttpBinding、WSFederationHttpBinding、NetTcpBinding、NetNamedPipeBinding
可靠的会话	WSHttpBinding、WSDualHttpBinding、WSFederationHttpBinding、NetTcpBinding
事务处理	WSHttpBinding、WSDualHttpBinding、WSFederationHttpBinding、NetTcpBinding、NetNamedPipeBinding、NetMsmqBinding、MsmqIntegrationBinding
双向通信	WSDualHttpBinding、NetTcpBinding、NetNamedPipeBinding、NetPeerTcpBinding

除了定义绑定之外，WCF 服务还必须定义端点。端点依赖于合约、服务的地址和绑定。例如，在下面的示例代码中，实例了一个 ServiceHost 对象，将地址“http://localhost:8080/MyWcfService”和一个 WSHttpBinding 实例绑定到服务的一个端点上。

```
static void StartService()
{
    ServiceHost host;
    Uri baseAddress = new Uri("http://localhost:8080/MyWcfService");
    host = new ServiceHost(typeof(Service1));
    WSHttpBinding binding = new WSHttpBinding();
    host.AddServiceEndpoint(typeof(Service1), binding, baseAddress);
    host.Open();
}
```

除了以编程方式定义绑定之外，还可以在应用程序的配置文件中定义它。WCF 的所有配置都位于<system.serviceModel>节点中，<service>节点定义了 WCF 中所提供的服务，<bindings>节点定义了绑定信息。

例如，下面的配置文件同样实现了上述代码的功能。

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="FirstWcfServiceLibrary.Service1">
        <host>
          <baseAddresses>
            <add baseAddress = "http://localhost:8080/MyWcfService" />
          </baseAddresses>
        </host>
        <endpoint address ="" binding="wsHttpBinding" contract=
        "FirstWcfServiceLibrary.IService1" bindingConfiguration=
        "config1"/>
      </service>
    </services>
    <bindings>
       <wsHttpBinding>
         <binding name="config1">
           <reliableSession enabled="true"/>
         </binding>
       </wsHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

可以看到，一个 WCF 服务必须要有一个端点，该端点包含地址、绑定和合约信息。WSHttpBinding 的默认配置由 bindingConfiguration 属性指定，该属性引用了下方名为“config1”的绑定配置信息。该配置信息位于<bindings>节点中，并启用了 reliableSession。

16.3.3　合约

任何一个分布式应用程序，之所以能够互相传递消息，都是事先制定好数据交换规则的，这个规则正是交换数据的双方（比如服务器端和客户端）能彼此理解对方的依据。WCF 作为分布式开发技术的一种，同样具有这样一种特性。而在 WCF 中制定的规则就被称为合约（Contract），它是 WCF 的消息标准，是任何一个 WCF 程序不可或缺的一部分。

在 WCF 中合约分为 4 种，分别为：定义服务操作的服务合约（Service Contract），定义自定义数据结构的数据合约（Data Contract），定义错误异常的异常合约（Fault Contract），以及直接控制消息格式的消息合约（Message Contract）。

1．服务合约

一般情况下，用接口（Interface）来定义服务合约。虽然也可以使用类（Class）来定义，但使用接口的好处更明显一些。主要表现在如下方面。

（1）便于合约的继承，不同的类型可以自由实现相同的合约。

（2）同一服务类型可以实现多个合约。

（3）和接口隔离原则相同，随时可以修改服务类型。

（4）便于制定版本升级策略，让新老版本的服务合约同时使用。

服务合约定义了 WCF 服务可以执行的操作，它包括 ServiceContract 和 OperationContract 两种。ServiceContract 用于类或者接口上，用于指定此类或者接口能够被远程调用，而 OperationContract 用于类中的方法上，用于指定该方法可被远程调用。

例如，下面的示例代码使用 ServiceContract 属性声明接口 IMyFirstService 可以被远程调用，OperationContract 属性声明 getTime()方法也可以被远程调用。

```
[ServiceContract]
public interface IMyFirstService
{
    [OperationContract]
    string getTime();
}
```

在表 16-4 中列出了 ServiceContract 属性的可用选项及其说明。

表 16-4　ServiceContract 属性的选项及说明

选项	说明
ConfigurationName	用于定义配置文件中服务配置的名称
CallbackContract	当服务用于双向消息传递时，此选项定义了客户端程序中实现的合约
Name	此选项定义了 WSDL 中 portType 节点的名称
Namespace	此选项定义了 WSDL 中 portType 节点的命名空间
SessionMode	此选项可定义调用这个合约的操作所需的会话。其值是 SessionMode 枚举值，可选的值有：Allowed、NotAllowed 和 Required
ProtectionLevel	此选项确定了绑定是否必须能保护通信。其值是 ProtectionLevel 枚举值，可选的值有：None、Sign 和 EntryptAndSign

在表 16-5 中列出了 OperationContract 属性的可用选项及其说明。

表 16-5 OperationContract 属性的选项及说明

选项	说明
Action	WCF 使用 SOAP 请求的 Action 选项把该请求映射到相应的方法上。因此使用此选项可以对请求的方法进行重命名
ReplyAction	此选项用于设置回应消息的 Action 名称
AsyncPattern	如果使用异步模式来实现操作，则可以将此选项设置为 true
IsInitiating	如果合约由一系列操作组成，则可以使用此选项指定初始化时执行的方法
IsTerminating	如果合约由一系列操作组成，则可以使用此选项指定结束时执行的方法
IsOneWay	使用此选项后，客户端程序将不会等待回应消息。因此在发送请求消息后，单向操作的调用者就无法直接检查是否失败
Name	操作的默认名称是方法的名称，使用此选项可进行重命名
ProtectionLevel	此选项可用于确定消息仅仅是签名，还是应先加密后签名

提 示

在服务合约中，还可以使用 DeliveryRequirements 属性定义服务的传输要求；使用 RequireOrderedDelivery 属性指定所传递的消息必须以相同的顺序到达；使用 QueuedDeliveryRequirements 属性指定消息以断开连接的方式传送。

2. 数据合约

数据合约也分为两种：DataContract 和 DataMember。DataContract 用于类或者结构上，指定此类或者接口能够被序列化并传输，而 DataMember 只能用在类或者接口的属性（Property）或者字段（Field）上，指定该属性或者字段能够被序列化传输。

数据合约的序列化不同于普通.NET 的序列化机制，在运行时所有的字段（包括私有字段）都会被序列化，而在执行数据合约的序列化时只有被标记了 DataMember 的属性才会被序列化。

例如，下面创建一个 Person 类并使用 DataContract 属性指定为可序列化。另外还创建了一个服务合约并定义了一个 Add()方法接收一个 Person 类型的参数。

```
[DataContract]
public class Person
{
    [DataMember]
    public int Id;
    [DataMember]
    public string Name;
    [DataMember]
    public DateTime Birthday;
    [DataMember]
    public string Email;
}
[ServiceContract]
```

```
public interface IPerson
{
    [OperationContract]
    bool Add(Person p);
}
```

如表 16-6 所示列出了 DataMember 属性可用的选项及其说明。

表 16-6　DataMember 属性的选项及说明

选项	说明
Name	序列化时默认名称与类中的声明相同，使用此选项可以进行重命名
Order	获取或设置成员的序列化和反序列化的顺序
IsRequired	获取或设置一个值，该值用于指定序列化引擎在读取或反序列化时成员必须存在
EmitDefaultValue	获取或设置一个值，该值指定是否对正在被序列化的字段或属性的默认值进行序列化。默认值为 true

例如，要对 Order 对象进行序列化，它的定义如下。在这里使用 Namespace 选项重新定义了命名空间。对于数据成员使用 Name 选项指定了一个别名，使用 Order 选项指定了显示的顺序。


```
[DataContract(Namespace = "http://www.itzcn.com")]
public class Order
{
    [DataMember(Name="OrderId",Order=1)]
    public Guid ID;
    [DataMember(Name="OrderDate",Order=2)]
    public DateTime Date;
    [DataMember]
    public string Customer;
    [DataMember]
    public string Address;
    public double TotalPrice;
}
```

执行后，Order 对象的序列化 XML 如下所示。

```
<Order xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:
//www.itzcn.com">
    <Address></Address>
    <Customer></Customer>
    <OrderId></OrderId>
    <OrderDate></OrderDate>
</Order>
```

通过定义的数据合约以及与最终生成 XML 结构的对比，可以总结出 WCF 默认采用如下的数据合约序列化规则。

（1）XML 的根节点为数据合约中类的名称，默认命名空间格式为 http://schemas.

datacontract.org/2004/07/{类所在命名空间}。

（2）只有使用 DataMember 属性定义的字段或者属性才能作为数据成员参与序列化（例如本实例中的 TotalPrice 属性不会再现在序列化后的 XML 中）。

（3）所有数据成员均以 XML 元素的形式被序列化。

（4）默认数据成员按照字母顺序排列。

（5）如果通过 Order 指定了顺序，且值相同，则以字母先后顺序排列。

（6）未指定 Order 的成员顺序在指定 Order 顺序之前。

（7）如果 DataContract 处于继承的类中，那么将优先显示父类中的成员。

3. 消息合约

如果需要在 WCF 服务中对 SOAP 消息进行控制则必须使用消息合约。在消息合约中，可以指定消息的哪些部分出现在 SOAP 标题中，哪些部分要放在 SOAP 的主体中。

例如，下面的示例演示了使用 ProcessOrderMessage 类定义消息合约的代码。

```
[MessageContract]
public class ProcessOrderMessage
{
    [MessageHeader]
    public Guid ID;
    [MessageBodyMember]
    public Order order;
}
```

如上述代码所示，在这里使用 MessageContract 属性指定 ProcessOrderMessage 为一个消息合约，对于 SOAP 消息中的标题使用 MessageHeader 属性指定，SOAP 主体使用 MessageBodyMember 属性指定。

为了使用上面定义的消息合约，这里将 IProcessOrder 接口定义为服务合约，并定义了一个可调用的 ProcessOrder()方法。代码如下所示。

```
[ServiceContract]
public interface IProcessOrder
{
    [OperationContract]
    ProcessOrderMessage ProcessOrder(ProcessOrderMessage message);
}
```

4. 异常合约

在 WCF 中所有合约基本上都是围绕着一个服务调用时的消息交换来进行的。例如，服务的客户端通过向服务的提供者发送请求消息；服务提供者在接收到该请求后激活服务实例，并调用相应的服务操作；最终将返回的结果以回复消息的方式返回给服务的客户端。

但是，如果服务操作不能正确地执行，服务端将会通过一种特殊的消息将错误信息

返回给客户端，这种消息被称为异常消息。对于异常消息，同样需要相应的合约来定义其结构，这种合约称为异常合约（Fault Contract）。

WCF 通过 FaultContract 属性来定义异常合约。由于异常合约是基于服务操作级别的，所以该属性将直接应用于服务合约接口或者操作合约的方法上。

下面的示例代码演示了 FaultContract 定义异常合约的方式。

```
[ServiceContract]
public interface IUser
{
    [OperationContract]
    [FaultContract(typeof(LoginTimeOut))]
    bool Login(string username, string userpass);
}
```

在上述代码中，使用 FaultContract 属性声明调用 Login()方法会抛出 LoginTimeOut 类的异常表示登录超时。

与本节前面介绍的其他合约一样，异常合约的 FaultContract 属性也有很多可用选项，如表 16-7 中列出了它们及其说明。

表 16-7 FaultContract 属性的选项及说明

选项	说明
Action	此选项用于设置当操作合约出现 SOAP 异常消息时要调用的操作。默认值为"当前操作的名称+Fault"
DetailType	此选项用于指定封装异常信息的自定义类，例如在上面定义的登录超时类 LoginTimeOut
Name	此选项用于设置 WSDL 中异常消息的名称
Namespace	此选项用于设置 SOAP 异常的命名空间
HasProtectionLevel	此选项用于设置 SOAP 异常消息是否分配有保护级别
ProtectionLevel	此选项用于设置 SOAP 异常消息要绑定的保护级别

16.4 端点

每个 WCF 服务都会关联到一个用于定位服务位置的地址，一个用于定义如何与服务进行通信的绑定，以及一个告知客户端服务能做什么的合约。这三样共同组成了服务的端点。

每个端点都必须完整拥有这三个组成部分，主机通过公开端点来对外提供服务。理论上，端点就是服务的外部交互接口，就像 CLR 或者 COM 接口。每个服务至少需要公开一个端点，服务上所有的端点都必须拥有唯一的定位地址，单个服务可以提供多个端点供不同类型的客户端调用。这些端点可以使用相同或不同的绑定对象，可以拥有相同或不同的服务契约。但是对于单个服务的不同端点，它们之间没有任何关联。

下面对如何通过配置文件和编程方式定义端点进行介绍。

16.4.1 通过配置文件方式

这种方式是将端点的信息保存到主机的配置文件中，通常是 app.config 文件或者 web.config 文件。

假设使用如下代码定义了一个服务合约及其实现类。

```
//指定命名空间
namespace MyNameSpace
{
    [ServiceContract]
    public interface IMyContract
    {
        //这里是 IMyContract 接口的定义
    }
    public class MyService : IMyContract
    {
        //这里是 IMyContract 接口的实现
    }
}
```

如下给出了针对这个 WCF 服务的端点信息，其中包含服务的完整名称、绑定类型、合约的完整名称等。

```
<system.serviceModel>
    <services>
        <service name = "MyNameSpace.MyService">
            <endpoint
                address  = "http://localhost:8000/MyService/"
                binding  = "wsHttpBinding"
                contract = "MyNameSpace.IMyContract"
            />
        </service>
    </services>
</system.serviceModel>
```

注意 *在这里指定的服务名称和服务合约必须是带命名空间的完整名称，否则将无法正确引用其地址。而且地址的类型与绑定类型必须匹配，否则就会在加载服务时导致异常。*

当然，也可以在配置文件中为一个单独的服务提供多个端点设置。这些端点可以使用相同的绑定类型，但是必须保证 URL 是唯一的。例如，如下是多个端点的示例配置文件。

```
<service name = "MyService">
```

```
        <endpoint
            address  = "http://localhost:8000/MyService/"
            binding  = "wsHttpBinding"
            contract = "IMyContract"
        />
        <endpoint
            address  = "net.tcp://localhost:8001/MyService/"
            binding  = "netTcpBinding"
            contract = "IMyContract"
        />
        <endpoint
            address  = "net.tcp://localhost:8002/MyService/"
            binding  = "netTcpBinding"
            contract = "IMyOtherContract"
        />
    </service>
```

还可以提供一个或多个默认的基本地址（Base Address），这样在端点设置中只需提供相对地址。多个基本地址之间不能冲突，不能在同一个端口进行监听。

相对地址通过端点绑定类型与基本地址进行匹配，从而在运行时获得完整地址。如果将某个端点设置中的地址设为空值（省略 address），则表示直接使用某个相匹配的基本地址。

例如，如下的配置文件演示了这种方式。

```
    <service name = "MyService">
        <host>
            <baseAddresses>
                <add baseAddress="http://localhost:8080/" />
                <add baseAddress="net.tcp://localhost:8081/" />
            </baseAddresses>
            </host>
            <endpoint
            address  = "MyService"       <!-- http://localhost:8080/
            MyService -->
            binding  = "wsHttpBinding"
            contract = "IMyContract"
        />
        <endpoint
            address  = "MyService"       <!-- net.tcp://localhost:8081/
            MyService -->
            binding  = "netTcpBinding"
            contract = "IMyContract"
        />
        <endpoint
            address  = "net.tcp://localhost:8002/MyService/"
            binding  = "netTcpBinding"
            contract = "IMyOtherContract"
```

```
    />
</service>
```

此外，还可以进一步对端点中的绑定参数进行设置。每种绑定类型可拥有多个名称不同的参数设置，然后在端点的 bindingConfiguration 属性中指定关联设置名称。

例如，在下面的配置文件中使用 bindingConfiguration 属性指定关联名称为“TransactionalTCP”的绑定信息。

```
<system.serviceModel>
    <services>
        <service name = "MyService">
            <endpoint
                address  = "net.tcp://localhost:8000/MyService/"
                bindingConfiguration = "TransactionalTCP"
                binding  = "netTcpBinding"
                contract = "IMyContract" />
        </service>
    </services>
    <bindings>
        <netTcpBinding>
            <binding name = "TransactionalTCP"
                transactionFlow = "true"   />
        </netTcpBinding>
    </bindings>
</system.serviceModel>
```

16.4.2 通过编程方式

编程方式配置端点与使用配置文件的方式是等效的。它的优点是不需要编写额外的配置文件，而是通过编程的方式将端点添加到 ServiceHost 实例中。如下所示为创建 ServiceHost 实例时可用的两个构造函数形式。

```
public ServiceHost(object singletonInstance, params Uri[]
baseAddresses);
public ServiceHost(Type serviceType, params Uri[] baseAddresses);
```

ServiceHost 提供了一个 AddServiceEndpoint()方法来向当前的服务中添加端点。如下所示为该方法的重载形式。

```
public ServiceEndpoint AddServiceEndpoint(Type implementedContract,
Binding binding, string address);
public ServiceEndpoint AddServiceEndpoint(Type implementedContract,
Binding binding, Uri address);
public ServiceEndpoint AddServiceEndpoint(Type implementedContract,
Binding binding, string address, Uri listenUri);
public ServiceEndpoint AddServiceEndpoint(Type implementedContract,
Binding binding, Uri address, Uri listenUri);
```

在使用时 address 参数可以是相对地址，也可以是绝对地址，这与使用配置文件时是一致的。例如，下面的代码演示了如何通过编程方式配置端点。

```
ServiceHost host = new ServiceHost(typeof(MyService));

Binding wsBinding  = new WSHttpBinding( );
Binding tcpBinding = new NetTcpBinding( );

host.AddServiceEndpoint(typeof(IMyContract), wsBinding, "http://
localhost:8000/MyService");
host.AddServiceEndpoint(typeof(IMyContract), tcpBinding, "net.tcp://
localhost:8001/MyService");
host.AddServiceEndpoint(typeof(IMyOtherContract), tcpBinding, "net.tcp:
//localhost:8002/MyService");

host.Open( );
```

16.5 实验指导——实现防盗链

很多时候，网站管理员不希望本网站的图片和文件被外部网站直接引用，这就需要实现防盗链。在 ASP.NET 中使用 IHttpHandler 接口可以非常简单地实现这个功能。IHttpHandler 接口是 HTTP 请求的处理中心，用于对请求页面进行真正的处理。处理的是 ASP.NET 注册过的文件类型（如 aspx 和 asmx），再根据请求处理后输出对应的内容。

为了实现防盗链需要自定义一个实现 IHttpHandler 接口的类，而且必须实现该接口中的 IsReusable 属性和 ProcessRequest()方法。IsReusable 属性表示获取一个值，该值指示其他请求是否可以使用 IHttpHandler 实例；ProcessRequest()方法则表示通过实现 IHttpHandler 接口的自定义 HttpHandler 启用 HTTP Web 请求的处理。

在本案例中，如果正常访问则会显示 images 目录下的 1.jpg；如果直接引用 URL 则会显示 error.jpg。具体步骤如下。

（1）创建 Default.aspx 页面，向该页面中添加一个元素链接，它链接到一个 ChainHandler.ashx 文件。内容如下。

```
<form id="form1" runat="server">
   <div><a href="ChainHandler.ashx">图片</a></div>
</form>
```

（2）向当前网站中添加一个名为 ChainHandler.ashx 的一般处理程序，并向该文件的 ProcessRequest()方法中添加实现防盗的代码。代码如下。

```
public void ProcessRequest(HttpContext context)
{
   string picpath = context.Server.MapPath("~/images/1.jpg");
                                                    //正常显示图片
   string errorpic = context.Server.MapPath("~/images/error.jpg");
                                                    //防盗图片
```

```
        if (context.Request.UrlReferrer == null) {    //如果请求为空
            context.Response.Expires = 0;
                                          //设置客户端缓冲时间过期时间为 0，即立即过期
            context.Response.Clear();    //清空服务器端为此会话开启的输出缓存
            context.Response.ContentType = "image/jpg";  //设置输出文件类型
            context.Response.WriteFile(errorpic); //将请求文件写入到输出缓存中
            context.Response.End();        //将输出缓存中的信息传送到客户端
        } else {
            if (context.Request.UrlReferrer.Host != "localhost") {
                                                //如果不是本地引用，则是盗链本站图片
                /* 省略防盗代码，可参考 if 语句 */
            }else {          //如果本地网站引用图片，如果是则返回正确的图片
                context.Response.Expires = 0;
                                          //设置客户端缓冲时间过期时间为 0，即立即过期
                context.Response.Clear(); //清空服务器端为此会话开启的输出缓存
                context.Response.ContentType = "image/jpg";  //设置输出文件类型
                context.Response.WriteFile(picpath);//将请求文件写入到输出缓存中
                context.Response.End();   //将输出缓存中的信息传送到客户端
            }
        }
    }
```


在上述代码中首先声明两个变量，它们分别用于显示正常图片和防盗图片；接着判断请求是否为空，如果是则输入防盗图片；否则再次进行判断，判断是否是本地网站引用图片。如果不是本地引用图片，则是防盗本站图片；否则将图片输出传送到客户端。

（3）运行 Default.aspx 页面查看效果，如图 16-11 所示。

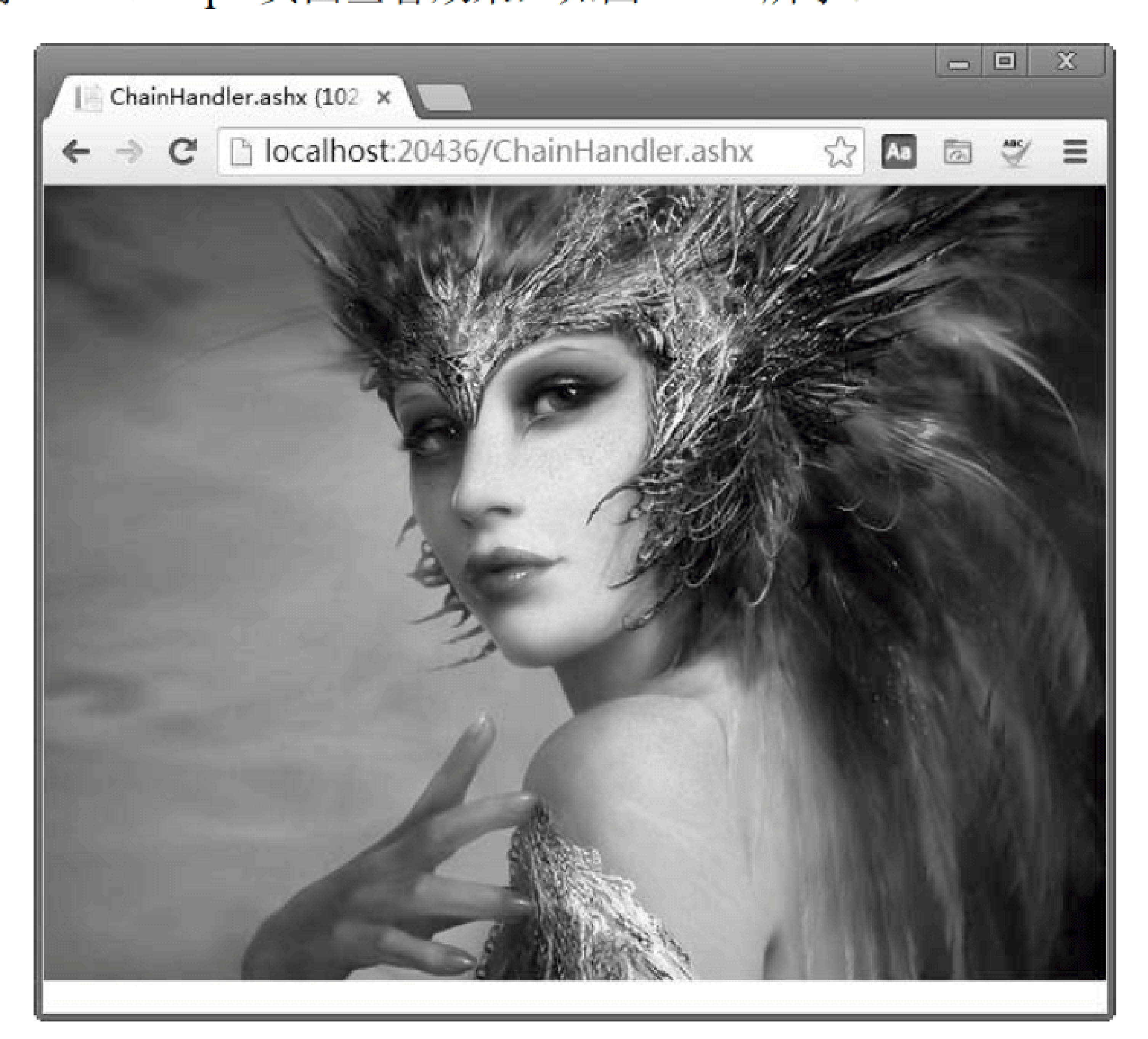


图 16-11　正常访问图片

（4）在浏览器中新建一个窗口输入相同的 URL 访问图片，此时会显示防盗链图片，如图 16-12 所示。



图 16–12 访问出错效果

思考与练习

一、填空题

1．WCF 是由__________开始引入，主要用于处理进程之间消息的传递。

2．在一个 WCF 服务中__________部分决定了服务的运行环境。

3．每个 WCF 端点都由绑定、__________和合约组成。

4．为了创建一个 WCF 服务，需要在类中使用__________属性声明服务合约。

5．__________接口是真正处理 HTTP 请求的核心。

二、选择题

1．下面关于 WCF 的描述，不正确的是__________。

A．WCF 是一个面向 Windows 编程的分布式架构

B．WCF 集成了 DCOM、Enterprise Service 以及 Web Service

C．使用 WCF 可以对服务进行定制、发布与运行

D．WCF 解决了跨平台特性

2．在 WCF 中使用来__________定义访问地址。

A．Contract

B．Binding

C．Address

D．EndPoint

3．WCF 的合约必须要以__________的方式定义。

A．类

B．接口

C．结构

D．数组

4．对于一个 WCF 地址，下面__________

选项是错误的。

A．http://localhost:2156/services/wcf

B．net.pipe:// localhost:2156/services/wcf

C．ip://192.168.0.135:2156/services/wcf

D．net.tcp:// localhost:2156/services/wcf

5．下列给出的绑定类型中，__________类型不是使用二进制编码进行通信。

A．NetMsmqBinding

B．NetTcpBinding

C．NetMsmqBinding

D．WSHttpBinding

三、简答题

1．简述 WCF 的作用及其重要组成部分。

2．简述 WCF 项目的创建及测试过程。

3．举例说明调用 WCF 服务的方法。

4．简述地址在 WCF 中的作用。

5．简述配置文件中定义端点的方法。

第 17 章　配置和部署 ASP.NET 网站

在本章之前已经详细介绍了 ASP.NET 开发网站所需的各种知识，但是这里会出现一个问题：创建网站并且实现功能确保无误后，如何让局域网内的其他用户能够访问该用户中的页面呢？很简单，只要用户将其网站进行部署和发布即可，本章将介绍最常用的三种部署方式。

在部署网站之前，还需要对网站的配置信息进行更改，例如，将连接字符串信息放置在 Web.config 文件中，或者在 Web.config 文件中配置其他信息等。因此，本章还将介绍与配置文件相关的信息。

通过本章的学习，读者不仅可以了解如何在 Web.config 文件中进行配置，也可以掌握如何部署和发布网站。

本章学习要点：

- ❑ 熟悉 Web.config 配置文件的基本结构
- ❑ 掌握如何创建一个 Web.config 配置文件
- ❑ 了解<appSettings>、<configSections>和<location>配置节
- ❑ 掌握<connectionStrings>节点的使用
- ❑ 掌握<system.web>配置节下的常用节点
- ❑ 掌握如何通过“发布网站”的方式发布网站
- ❑ 熟悉“复制网站”和 XCOPY 发布网站

17.1　了解配置文件

.NET Framework 提供的配置管理包括范围广泛的设置，允许管理员管理 Web 应用程序及其环境。这些设置存储在 XML 配置文件中，其中一些控制计算机范围的设置，而另一些控制应用程序特定的配置。

ASP.NET 的配置文件为 XML 文件，.NET Framework 定义了实现配置设置的一系列元素，并且 ASP.NET 配置架构包含控制 ASP.NET Web 应用程序的行为的元素。本节将简单介绍一下 ASP.NET 中的配置文件。

17.1.1　配置文件概述

程序开发人员可以使用任何文本编辑器编辑配置文件，它实际上是遵循 XML 文档格式的。在 ASP.NET Web 应用程序的配置文件中，Web.config 文件最为常用，但是，除

了该文件外，还有用于其他功能的配置文件。例如，配置文件可以用于应用程序，也可以用于机器和代码访问安全性。如图 17-1 所示为可用于配置 ASP.NET Web 应用程序的配置文件。

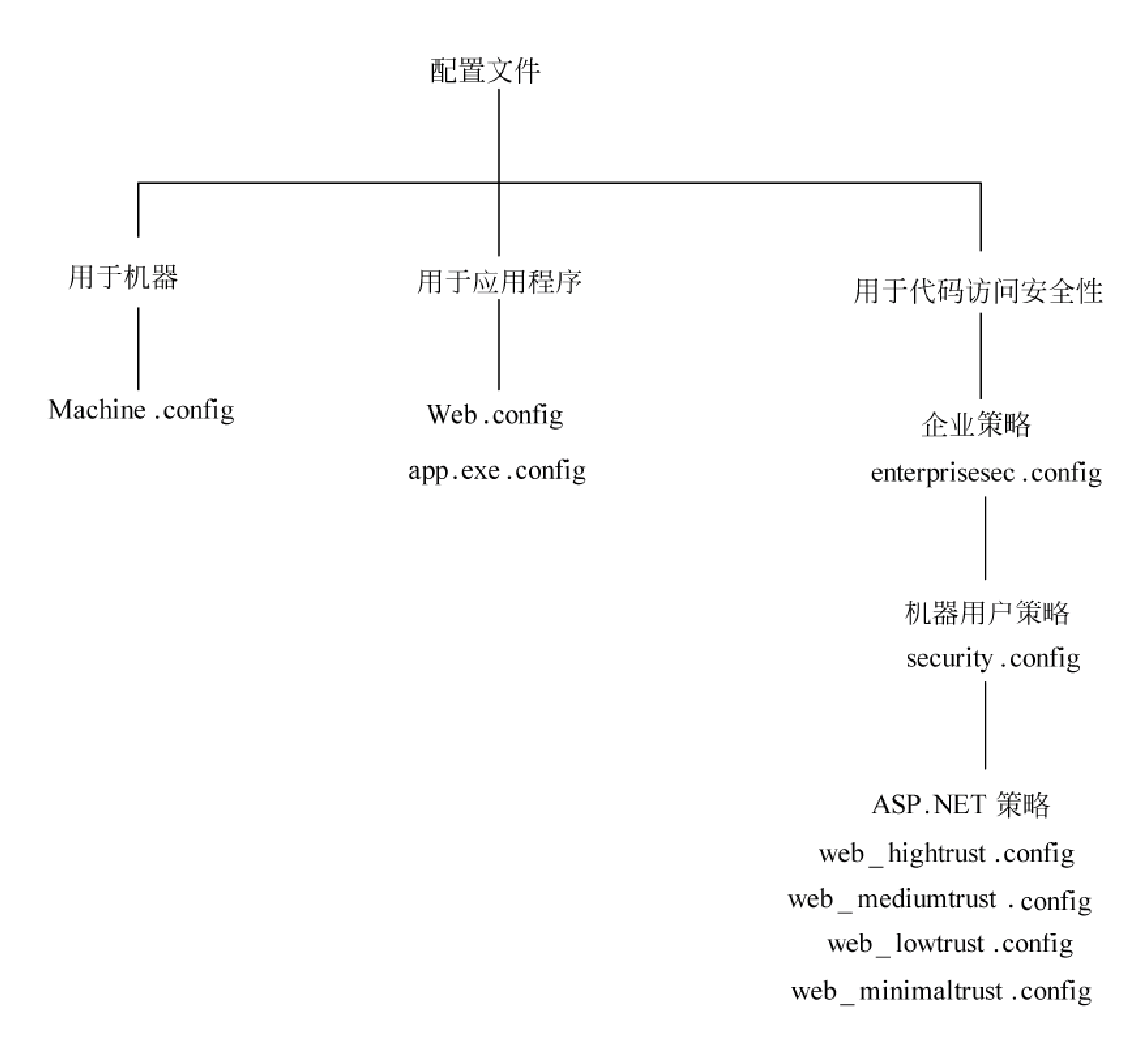


图 17-1 配置 ASP.NET Web 应用程序的配置文件

从图 17-1 中可以看出：配置 Web 应用程序的配置文件可以有多种，Machine.config 和 Web.config 文件共享许多相同的配置部分和 XML 元素。另外在该图中，Machine.config 文件用于将计算机访问的策略应用到本地计算机上运行的所有.NET Framework 应用程序，开发者还可以使用应用程序特定的 Web.config 文件自定义单个应用程序的设置。

在图 17-1 中的 Machine.config 文件包含整个服务器的配置信息；Web.config 文件包含运行一个 ASP.NET 服务器需要的所有配置信息。

17.1.2 配置文件及其说明

根据图 17-1 中可以看出，用于不同功能的配置文件包含多个。开发者对配置文件所做的更改将会被动态应用，通常无须重新启动服务器或者任何服务，除非更改了 Machine.config 中的名称为 processModel 的节点元素。

例如，表 17-1 对图 17-1 中的文件进行了简单说明，并且显示了这些配置文件的位置。

表 17-1　配置文件的说明及其位置

配置文件	说明	说明
Machine.config	每台计算机每个.NET Framework 安装版一个	%system32%\Microsoft.NET\Framework\{version}\CONFIG 以 Windows XP 操作系统为例，Machine.config 文件存储在：C:\WINDOWS\Microsoft.NET\ Framework\[version]\Config
Web.config	每个应用程序有零个、一个或多个	\inetpub\wwwroot\Web.config \inetpub\wwwroot\YourApplication\Web.config \inetpub\wwwroot\YourApplication\SubDir\Web.config
Enterprisesec.config	企业级 CAS 配置	%system32%\Microsoft.NET\Framework\{version}\CONFIG
Security.config	计算机级 CAS 配置	%system32%\Microsoft.NET\Framework\{version}\CONFIG
	用户级 CAS 配置	\Documents and Settings\{user}\Application Data\Microsoft\CLR Security Config\{version}
Web_hightrust.config Web_mediumtrust.config Web_lowtrust.config Web_minimaltrust.config	ASP.NET Web 应用程序 CAS 配置	%system32%\Microsoft.NET\Framework\{version}\CONFIG

17.1.3　配置文件的保存和加载

配置文件有很多，例如配置企业级 CAS 的 Enterprisesec.config 文件、计算机级 CAS 的 Security.config 文件和 Web_hightrust.config 等。但是，最主要的配置文件有两个：Machine.config 和 Web.config。这两个文件一般都不需要开发人员手工去维护，直接保持默认的内容即可。

但是，针对 ASP.NET 应用程序而言，它自身会包含 0 个、1 个或者多个 Web.config 配置文件。ASP.NET 网站 IIS 启动的时候会加载这些配置文件中的配置信息，然后缓存这些信息，这样就不必每次都去读取配置信息。在运行过程中，ASP.NET 应用程序会监视配置文件的变化情况，一旦编辑了这些配置信息，就会重新读取这些配置信息并且缓存。

如下所示为 ASP.NET 网站运行时加载配置文件的顺序。

（1）如果在当前网页所在的目录下存在 Web.config 文件，则查看是否存在所要查找的节点名称。如果存在则返回结果，并且停止查找。

（2）如果当前网页所在的目录下不存在 Web.config 文件，或者此文件中不存在该节点名，则查找它的上一级目录，直到网站的根目录。

（3）如果网站根目录下不存在 Web.config 文件，或者此文件中不存在该节点名，则转到“Windows 目录\Microsoft.NET\Framework\对应.net 版本\config\Web.config”中进行查找。

（4）如果在上一条的文件夹下还没有找到相应节点，则在“Windows 目录\Microsoft.NET\Framework\对应.net 版本\config\machine.config”中进行查找。

（5）如果仍然没有找到相应的节点，则返回 null。

通常情况下，开发人员对某个网站或者某个文件夹有特定的要求配置时，可以在相应的文件夹下创建一个 Web.config 文件，这样会覆盖一级文件夹 Web.config 文件中的同名配置文件，这些配置信息只查找一次，以后就会被缓存起来供后来的调用。

ASP.NET 应用程序运行过程中，Web.config 文件发生更改就会导致相应的应用程序重新启动，这时存储在服务器内存中的用户会话信息就会丢失（如存储在内存中的 Session）。一些软件（如杀毒软件）每次完成对 Web.config 的访问时就会修改 Web.config 的访问时间属性，也会导致 ASP.NET 应用程序的重启。

17.2 了解 Web.config 文件

在众多的配置文件中 Web.config 是最常用到的配置文件，它用于设置应用程序的配置信息。下面将详细介绍 Web.config 配置文件的优点、创建方法、文件的结构以及其中常用的配置节点等。

17.2.1 Web.config 文件的优点

Web.config 文件使 ASP.NET 应用程序的配置变得灵活、高效和容易实现，同时 Web.config 配置文件还为 ASP.NET 应用提供了可扩展的配置，使得应用程序能够自定义配置。

Web.config 配置文件主要具有如下优点。

1．配置设置易读性

由于 Web.config 配置文件是基于 XML 文件类型，所有的配置信息都存放在 XML 文本文件中，可以使用文本编辑器或者 XML 编辑器直接修改和设置相应配置节，相比之下，也可以使用记事本进行快速配置而无须担心文件类型。

2．更新的即时性

在 Web.config 配置文件中某些配置节被更改后，无须重启 Web 应用程序就可以自动更新 ASP.NET 应用程序配置。但是在更改有些特定的配置节时，Web 应用程序会自动保存设置并重启。

3．本地服务器访问

在更改了 Web.config 配置文件后，ASP.NET 应用程序可以自动探测到 Web.config 配置文件中的变化，然后创建一个新的应用程序实例。当用户访问 ASP.NET 应用程序时会被重定向到新的应用程序。

4．安全性

由于 Web.config 配置文件通常存储的是 ASP.NET 应用程序的配置，所以 Web.config

配置文件具有较高的安全性，一般的外部用户无法访问和下载 Web.config 配置文件。当外部用户尝试访问 Web.config 配置文件时，会导致访问错误。

5. 可扩展性

Web.config 配置文件具有很强的扩展性，通过 Web.config 配置文件开发人员能够自定义配置节，在应用程序中自行使用。

6. 保密性

开发人员可以对 Web.config 配置文件进行加密操作而不会影响配置文件中的配置信息。虽然 Web.config 配置文件具有安全性，但是通过下载工具依旧可以进行文件下载，对 Web.config 配置文件进行加密，可以提高应用程序配置的安全性。

17.2.2 创建 Web.config 文件

所有的.config 文件实际上都是一个 XML 文本文件，Web.config 文件也不例外。它可以出现在应用程序的每一个目录中。当创建一个 Web 应用程序后，默认情况下会自动创建一个默认的 Web.config 文件，包括默认的配置设置，所有的子目录都继承它的配置设置，如果想要修改子目录的配置设置，可以在该子目录下新建一个 Web.config 文件，它可以提供除从父目录继承的配置信息以外的配置信息，也可以重写或修改父目录中定义的设置。

【范例 1】

例如，当用户新建一个默认的 ASP.NET 空网站时，自动创建的 Web.config 文件的配置信息如下。

```
<?xml version="1.0"?>
<!--
  For more information on how to configure your ASP.NET application, please
  visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
    <system.web>
      <compilation debug="false" targetFramework="4.0" />
    </system.web>
</configuration>
```

上述信息非常简单，configuration 是节的根元素，其他节都是在它的内部；在该元素下添加一个 system.web 节点，它用于控制 ASP.NT 运行时的行为。在 system.web 节点下创建 compilation 子节点，compilation 中的 debug 属性指向是否调试程序；targetFramework 指向.NET Framework 的版本。

一个应用程序中可以包含一个或者多个 Web.config 文件，因此，用户可以手动创建一个 Web.config 文件，并且向该文件中自定义配置信息。

【范例 2】

新建一个 Web.config 配置文件时很简单，一种是直接复制根目录中存在的 Web.config 文件到指定的目录下，在复制后的文件中自定义配置信息；另一种是像创建 Web 窗体页那样创建一个 Web.config 配置文件。

创建 Web.config 配置文件的主要步骤是：选择当前项目后右击，选择【添加新项】菜单项，这时会弹出一个【添加新项】对话框。在弹出的对话框中找到【Web 配置文件】项并选中，选中后直接单击【添加】按钮添加，效果如图 17-2 所示。

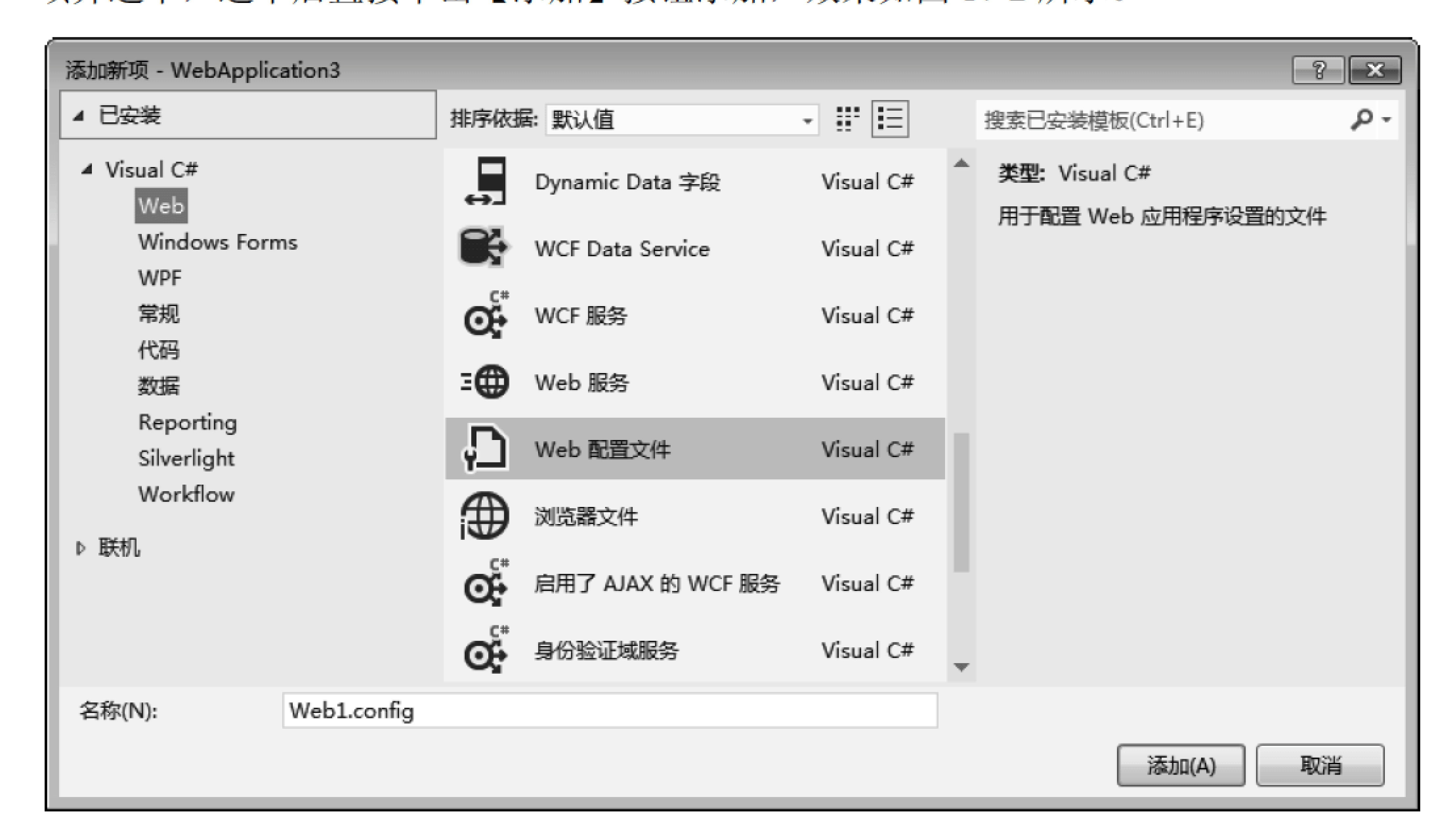


图 17-2 创建一个新的 Web.config 配置文件

17.2.3 配置文件结构

所有的 ASP.NET 配置信息都驻留在.config 文件中，位于.NET Framework 系统文件夹中的 Machine.config 文件和根 Web.config 文件为服务器上运行的所有网站提供默认值。位于网站的根文件夹中的每个网站的 Web.config 文件提供对这些值的网站特定的重写，可以将其他 Web.config 文件置于一个网站的子文件夹中，以提供专用于该网站内的文件夹的重写。

Web.config 文件包含将 configuration 元素作为根节点的 XML，此元素中的信息分为两个主区域：配置节处理程序声明区域和配置节设置区域。

【范例 3】

节处理程序是用来实现 ConfigurationSectiono 接口的.NET Framework 类，声明区域可标识每个节处理程序类的命名空间和类名。节处理程序用于读取和设置与节有关的设置，下面的代码演示 Web.config 文件的 XML 结构的简化视图。

```
<configuration>
```

```
    <!-- Configuration section-handler declaration area. -->
    <configSections>
      <section name="section1" type="section1Handler" />
      <section name="section2" type="section2Handler" />
    </configSections>
    <!-- Configuration section settings area. -->
    <section1>
      <s1Setting1 attribute1="attr1" />
    </section1>
    <section2>
      <s2Setting1 attribute1="attr1" />
    </section2>
    <system.web>
      <authentication mode="Windows" />
    </system.web>
  </configuration>
```

在 Web.config 文件中，如果在位于配置层次结构中更高级别的.config 文件中声明节，则相应的节会在声明区域中尚未声明的设置区域中出现。例如，Machine.config 文件中声明了 system.web 节点。因此，无须在单个网站级 Web.config 文件中声明此节，如果在位于网站的根文件夹的 Web.config 文件中声明某个节，则可以包含该节的设置，而无须将其包含在位于子文件夹中的 Web.config 文件中。

1．配置节处理程序声明

配置节处理程序声明区域位于配置文件中的 configSections 元素内，它包含在其中声明节处理程序的 ASP.NET 配置 section 元素。可以将这些配置节处理程序声明嵌套在 sectionGroup 元素中，以帮助组织配置信息。通常，sectionGroup 元素表示要应用配置设置的命名空间。例如，所有 ASP.NET 配置节处理程序都分组到 system.web 节组中。代码如下。

```
<sectionGroup name="system.web"
    type="System.Web.Configuration.SystemWebSectionGroup, System.Web,
    Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">
    <!-- <section /> elements. -->
</sectionGroup>
```

配置节设置区域中的每个配置节都有一个节处理程序声明，节处理程序声明中包含配置设置节的名称（如 pages）以及用来处理该节中配置数据的节处理程序类的名称（如 System.Web.Configuration.PagesSection）。下面的代码演示映射到配置设置节的节处理程序类。

```
<section name="pages"
    type="System.Web.Configuration.PagesSection, System.Web, Version=
    2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">
</section>
```

程序员可需要声明配置节处理程序一次，默认 ASP.NET 配置节的节处理程序已经在 Machine.config 文件中进行了声明。网站的 Web.config 文件和 ASP.NET 应用程序中的其他配置文件都自动继承在 Machine.config 文件中声明的配置处理程序。只有当创建用来处理自定义设置节的自定义节处理程序类时，才需要声明新的节处理程序。

2. 配置节设置

配置节设置区域位于配置节处理程序声明区域之后，它包含实际的配置设置。默认情况下，在当前配置文件或某个根配置文件中，对于 configSections 区域中的每个 section 和 sectionGroup 元素都会有一个配置节元素。可以在 systemroot\Microsoft.NET\Framework\versionNumber\CONFIG\Machine.config.comments 文件中查看这些默认值。

【范例 4】

配置节元素还可以包含子元素，这些子元素与其父元素由同一个节处理程序处理。例如，下面的 pages 元素包含一个 namespace 元素，该元素没有相应的节处理程序，这是因为它是由 pages 节处理程序来处理的。

```
<pages buffer="true" enableSessionState="true" asyncTimeout="45"
  <!-- Other attributes. -->
>
  <namespaces>
    <add namespace="System" />
    <add namespace="System.Collections" />
  </namespaces>
</pages>
```

17.2.4 Web.config 的常用配置节

可以向 Web.config 文件中自定义配置信息，下面列出了一些常用的配置节以及配置信息。

1. <appSettings>配置节

<appSettings>配置节用于定义应用程序设置项，对一些不确定的设置，还可以让用户根据自己的实际情况进行设置。向该配置节中添加内容时需要通过<add>节点，它常用的属性有两个，分别是 key 属性和 value 属性。其中，key 属性指定自定义属性的关键字，以方便在应用程序中使用该配置节；value 属性表示自定义属性的值。细心的读者可以发现，在介绍模块和处理程序时使用过<appSettings>配置节。

【范例 5】

下面向<appSettings>配置节中添加两个<add>节点内容。代码如下。

```
<appSettings>
  <add key="ConnectionStr" value="server=.;userid=sa;password=123456;
  database=mytest;" />
```

```
    <add key="ErrPage" value="Error.aspx" />
  </appSettings>
```

上述代码第一个<add>节点定义了一个连接字符串常量，并且在实际应用时可以修改连接字符串，不用修改程序代码；第二个<add>节点定义了一个错误重定向页面，它指向 Error.aspx 页面。

一般情况下，向<appSettings>配置节下添加内容后可以在页面中进行调用，以范例 5 中 ConnectionStr 为例，它可以在通用类中进行获取。通过 ConfigurationManager 对象的 AppSettings[int index]或者 AppSettings[string key]属性获取对应 key 的 value 属性值。其中，index 表示在配置节中的索引值；而 key 则表示配置节中的 key 值。

【范例 6】

下面在后台页面中分别通过索引和 key 两种形式获取连接字符串的内容，这两种形式的效果是一样的。代码如下。

```
protected void Page Load(object sender, EventArgs e)
{
    string conn = ConfigurationSettings.AppSettings[0].ToString();
    string conn = ConfigurationSettings.AppSettings["ConnectionStr"].
    ToString();
}
```


2. <configSections>配置节

<configSections>指定了配置节和处理程序声明，虽然 ASP.NET 不对如何处理配置文件内的设置做任何假设，但是 ASP.NET 会将配置数据的处理委托给配置节处理程序。该配置节由多个<section>配置节组成，可以将这些配置节处理程序声明嵌套在<sectionGroup>节点中，以帮助组织配置信息。

<configSections>配置节的组成结构如下。

```
<configSections>
  <section />
  <sectionGroup></sectionGroup>
</configSections>
```

上述结构中，<section/>定义配置节处理程序与配置元素之间的关联；<sectionGroup/>定义配置节处理程序与配置节之间的关联。可以在<sectionGroup>中对<section>进行逻辑分组，以对<section>进行组织，并且避免命名冲突。

【范例 7】

<sectionGroup>中包含 name 和 type 两个属性，而<section>中这两个属性也经常被使用到。其中，name 属性指定配置数据配置节的名称；而 type 属性指定与 name 属性相关的配置处理程序类。在本例子中首先向<configSections>中自定义配置信息，然后在页面中进行调用。操作步骤如下。

（1）向 Web.config 配置文件的在<configuration>根节点下添加一个<configSections>子节点，它必须是根元素下的第一个子元素节点。代码如下。

```
<configSections>
  <sectionGroup name="mySectionGroup">
    <section name="mySection" requirePermission="true" type=
    "SectionHandler" />
  </sectionGroup>
</configSections>
```

（2）在步骤（1）中<sectionGroup>节点中提到了 mySectionGroup 和 mySection，它们是在 Web.config 中自定义的模块。代码如下。

```
<mySectionGroup>
  <mySection>
    <add key="No1" value="李磊" />
    <add key="No2" value="许飞" />
    <add key="No3" value="陈艳" />
    <add key="No4" value="朱小小" />
    <add key="No5" value="陈海风" />
  </mySection>
</mySectionGroup>
```

（3）向当前网站下创建一个 SectionHandler 类，该类实现 IConfigurationSectionHandler 接口。在该类中为 Create()方法添加代码，在代码中创建 Hashtable 集合类，向该集合对象中读取数据并添加。代码如下。

```
public class SectionHandler : IConfigurationSectionHandler
{
    public object Create(object parent, object configContext, System.Xml.
    XmlNode section) {
        Hashtable ht = new Hashtable();                //创建 Hashtable 数据
        foreach (XmlNode node in section.ChildNodes) {
            if (node.Name == "add")
                ht.Add(node.Attributes["key"].Value, node.Attributes
               ["value"].Value);
        }
        return ht;
    }
}
```

（4）向 Web 窗体页面的后台 Load 事件中添加代码，首先通过 ConfigurationManager 类的 GetSection()方法获取 Web.config 配置文件中 mySectionGroup/mySection 元素节点下的内容，得到返回的 Hashtable 对象。然后遍历 Hashtable 集合对象中的信息，并且将内容输出到网页中。代码如下。

```
protected void Page Load(object sender, EventArgs e)
{
    Hashtable ht = ConfigurationManager.GetSection("mySectionGroup/
    mySection") as Hashtable;
    foreach (DictionaryEntry de in ht) {    //遍历 Hashtable 对象
```

```
        Response.Write(de.Key + " - " + de.Value + "<br>");
    }
}
```

（5）运行本次例子的代码查看窗体页的输出结果，最终效果图不再显示。

3．<location>配置节

在 Web.config 配置文件中的<location>配置节也会被使用到，它可以在同一个配置文件中指定多个设定组，使用<location>元素的 path 属性可以指定设定应该被应用到的子目录或文件。

【范例 8】

多个<location>节点可以存在于同一个配置文件中，并且为相同的配置节指定不同的范围。例如，下面向 Web.config 配置文件中添加两个<location>节点，设置该节点的 path 属性，然后分别向<location>节点下添加内容，指定 HttpHandler 处理时的相关信息。代码如下。

```
<configuration>
   <system.web>
      <sessionState cookieless="true" timeout="10" />
   </system.web>
   <location path="sub1">   <!-- Configuration for the "Sub1" 
   subdirectory. -->
      <system.web>
         <httpHandlers>
            <add verb="*" path="Sub1.Scott" type="Sub1.Scott" />
            <add verb="*" path="Sub1.David" type="Sub1.David" />
         </httpHandlers>
      </system.web>
   </location>
   <location path="sub2">    <!-- Configuration for the "Sub2" 
   subdirectory. -->
      <system.web>
         <httpHandlers>
            <add verb="*" path="Sub2.Scott" type="Sub2.Scott" />
            <add verb="*" path="Sub2.David" type="Sub2.David" />
         </httpHandlers>
      </system.web>
   </location>
</configuration>
```

4．<connectionStrings>配置节

除了向<appSettings>节点中添加连接字符串的内容外，还可以向<connectionStrings>节点中添加数据库的连接字符串信息。代码如下。

```
<connectionStrings>
```

```
    <add name="ConnStr" connectionString="server=.;userid=sa;password=
    123456;database=mytest;"/>
</connectionStrings>
```

在上述代码中通过 name 指定连接字符串的名称；connectionString 属性来指定连接的字符串，包括数据库名称和连接用户名与密码等。

17.2.5 <system.web>配置节

除了 17.2.4 节介绍的三个配置节外，还有一个配置节被使用到，通过向该配置节中添加子节点不仅可以指定数据库连接字符串，也可以实现身份验证和自定义错误等。下面将详细介绍<system.web>配置节。

1. <authentication>节点

<authentication>节点用来配置 ASP.NET 身份验证方案，该方案用于识别查看 ASP.NET 应用程序的用户。此节点中最常用的属性是 mode，该属性包含以下 4 种身份验证模式。

1）Windows（默认验证）

将 Windows 验证指定为默认的身份验证模式，将它与以下任意形式的 Microsoft Internet 信息服务（IIS）身份验证结合起来使用：基本、摘要、集成 Windows 身份验证（NTLM/Kerberos）或证书。在这种情况下，开发人员的应用程序将身份验证责任委托给基础 IIS。

2）Forms 身份验证

将 ASP.NET 基于窗体的身份验证指定为默认身份验证模式，Forms 身份验证是最常用的一种身份验证方式。

3）Passport 身份验证

将 Microsoft Passport Network 身份验证指定为默认身份验证模式。

4）None

不会指定任何身份验证，允许匿名访问，或手动编码控制用户访问。

【范例 9】

例如，下面的代码为基于窗体（Forms）的身份验证配置站点，当没有登录的用户访问需要验证的网页，网页自动跳转到登录页面。代码如下。

```
<authentication mode="Forms" >
  <forms loginUrl="logon.aspx" name=".FormsAuthCookie"/>
</authentication>
```

上述代码中 loginUrl 表示登录网页的名称，即网址；name 则表示 Cookie 的名称。除了 loginUrl 和 name 属性外，还可以向该节点中添加其他属性，这些属性说明如下。

（1）name：指定用于身份验证的 Cookie 名称。

（2）loginUrl：指定为登录而要重定向的 URL，默认值为 Default.aspx。

（3）defaultUrl：默认页的 URL，通过 FormsAuthentication.DefaultUrl 属性得到该值。

（4）timeout：指定以整数分钟为单位，表单验证的有效时间即是 Cookie 的过期时间。

（5）path：Cookie 的作用路径。默认为正斜杠（/），表示该 Cookie 可用于整个站点。

（6）requireSSL：在进行 Forms 身份验证时，与服务器交互是否要求使用 SSL。

（7）slidingExpiration：是否启用“弹性过期时间”，如果该属性设置为 false，从首次验证之后过 timeout 时间后 Cookie 即过期；如果该属性为 true，则从上次请求开始过 timeout 时间才过期，这表示首次验证后，如果保证每 timeout 时间内至少发送一个请求，则 Cookie 将永远不会过期。

（8）enableCrossAppRedirects：是否可以将已进行了身份验证的用户重定向到其他应用程序中。

（9）cookieless：定义是否使用 Cookie 以及 Cookie 的行为。

（10）domain：Cookie 的域。

2. <authorization>节点

<authorization>节点的作用是控制对 URL 资源的客户端访问（例如允许匿名用户访问），该元素可以在任何级别（计算机、站点、应用程序、子目录或页）上声明，它必须与<authentication>节点配合使用。

【范例 10】

下面通过向<authorization>节点中添加内容，禁止匿名用户的访问，允许角色是 admin 的访问。代码如下。

```
<system.web>
    <authorization>
        <deny users="?"/>
        <allow roles="admin"/>
    </authorization>
</system.web>
```

上述代码<authorization>节点下添加 deny 和 allow 两个元素，其中 deny 表示拒绝，allow 表示允许。deny 和 allow 中都可以添加 user、roles 和 verbs 这三个属性，其说明如下。

（1）user：一个使用逗号进行分隔的用户名列表，列表中的用户被授予（或拒绝）对资源的访问。其中“？”表示匿名用户，而“*”则代表所有用户。

（2）roles：逗号进行分隔的角色列表，这些角色被授予（或拒绝）对资源的访问。

（3）verbs：逗号进行分隔的谓词列表，比如 GET、HEAD、POST 或 DEBUG 等。

3. <customErrors>节点

<customErrors>能够指定当出现错误时系统自动跳转到一个错误发生的页面，同时也能够为应用程序配置是否支持自定义错误。添加<customErrors>节点时可以指定它的 mode 属性和 defaultRedirect 属性，其中前者表示自定义错误的状态，后者表示应用程序发生错误时所要跳转的页面。mode 属性的取值有 On、Off 和 RemoteOnly 三个，说明

如下。

（1）On：表示启动自定义错误。

（2）Off：表示不启动自定义错误。

（3）RemoteOnly：表示给远程用户显示自定义错误。

【范例 11】

下面分别向<customErros>节点中添加两个<error>子节点，这两个节点用于处理访问页面时出现的 403 和 404 错误。代码如下。

```
<system.web>
  <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.
  htm">
    <error statusCode="403" redirect="NoAccess.htm" />
    <error statusCode="404" redirect="FileNotFound.htm" />
  </customErrors>
</system.web>
```

从上述代码中可以看出，添加<error>节点时为其指定 statusCode 属性和 redirect 属性，statusCode 属性用于捕捉发生错误时的状态码。例如，请求资源不可用时的 403 错误；找不到网页时的 404 错误；redirect 属性表示发生错误后跳转的页面。

4．<httpModules>节点和<httpHandlers>节点

细心的读者对这两个节点一定会很熟悉，在第 10 章中提到过模块和处理程序，它们就是用来配置与模块和处理程序有关的信息的。

【范例 12】

<httpModules>节点定义 HTTP 请求时与模块有关的信息。代码如下。

```
<httpModules>
    <add name="MyHttpModule" type="MyHttpModule"/>
</httpModules>
```

<httpHandlers>节点可以用于根据请求中指定的 URL 和 HTTP 谓词（如 GET 和 POST）将传入的请求映射到相应的处理程序，可以针对某个特定的目录下指定的特殊文件进行特殊处理。

【范例 13】

下面向<httpHandlers>节点添加内容，针对网站 path 目录下的所有*.jpg 文件来编写自定义的处理程序。代码如下。

```
<httpHandlers>
    <add path="path/*.jpg" verb="*" type="HttpHandlerImagePic"/>
</httpHandlers>
```

5．<httpRuntime>节点

<httpRuntime>节点的作用是配置 ASP.NET HTTP 运行库设置，该节可以在计算机、

站点、应用程序和子目录级别声明。

【范例 14】

下面向<httpRuntime>节点中添加内容，控制用户上传文件最大为 4MB，最长时间为 60s，最多请求数为 100。代码如下。

```
<system.web>
   <httpRuntime maxRequestLength="4096" executionTimeout="60"
   appRequestQueueLimit="100"/>
</system.web>
```

6. <sessionState>节点

<sessionState>节点的作用是为当前应用程序配置会话状态设置，例如设置是否启用会话状态和会话状态保存位置等。

【范例 15】

例如，下面向<sessionState>节点中添加代码，分别指定 mode、cookieliess 和 timeout 属性的值。代码如下。


```
<sessionState mode="InProc" cookieless="true" timeout="20"/>
</sessionState>
```

在上述代码中，mode 属性表示在本地储存会话状态；cookieless 属性的值表示如果用户浏览器不支持 Cookie 时启用会话状态（默认为 false）；timeout 表示会话可以处于空闲状态的分钟数。mode 属性的取值可以有多个，包括 Off、InProc、StateServer 和 SqlServer 这 4 个，说明如下。

（1）Off：设置为该值时表示禁用该设置。

（2）InProc：表示在本地保存会话状态。

（3）StateServer：表示在服务器上保存会话状态。

（4）SqlServer：表示在 SQL Server 保存会话设置。

除了上述介绍的向<sessionState>节点中添加的属性外，还可以添加其他的属性，例如用来指定远程存储会话状态的服务器名和端口号的 stateConnectionString 属性；用来连接 SQL Server 的连接字符串的 sqlConnectionString 属性，当在 mode 属性中设置 SqlServer 时，则需要使用到该属性。

7. <trace>节点

<trace>节点的作用是配置 ASP.NET 跟踪服务，主要用来程序测试判断哪里出错。下面的代码为 Web.config 中的默认配置。

```
<trace enabled="false" requestLimit="10" pageOutput="false" traceMode=
"SortByTime" localOnly="true" />
```

上述代码中 enabled 表示是否启用连接，默认值为 false 时表示不启用连接；requestLimit 属性表示指定在服务器上存储的跟踪请求的数目；pageOutput 属性设置为 false 时表示只能通过跟踪实用工具访问跟踪输出；traceMode 属性指定为 SortByTime，表示以处理跟踪的顺序来显示跟踪信息；localOnly 属性的值为 true，表示跟踪查看器（trace.axd）只用于宿主 Web 服务器。

17.3 网站部署和发布

部署是指将应用程序从开发环境移植到运行环境的过程。在部署网站之前需要进行两个操作：首先需要在 Web.config 配置文件中关闭调试功能，因为调试功能打开会降低应用程序的性能；其次需要使用 Release（发行版）方式编译应用程序（直接单击工具栏中的配置即可）。

下面通过两种方式介绍如何将一个开发好的网站部署到目标环境。

17.3.1 通过“发布网站”工具发布

“发布网站”工具可以将某些源代码编译为程序集，然后将这些程序集和其他必需的文件复制到指定的文件夹，可以使用任何所需的方法将文件复制到其他服务器。这种方式经常被使用到，通过这种方式发布的网站没有源代码，即不会显示.aspx.cs 文件及其内容。

“发布网站”工具对网站内容进行预编译，然后将输出复制到所指定的目录或服务器位置。使用文件传输协议（FTP）或 HTTP，可以将输出写入本地或内部网络文件系统中可用的任何文件夹中。必须具有相应权限才能向目标网站写入，可以在发布过程中直接发布到 Web 服务器，也可以预编译到本地文件夹，然后自己将文件复制到 Web 服务器。

“发布网站”工具通常适用于以下两种情况。

（1）希望预编译站点以避免将源代码或标记放在 Web 服务器上，这有助于保护知识产权。

（2）希望进行预编译，以避免 Web 服务器首次请求某页时由动态编译引发的延迟。在决定此原因需要预编译站点之前，应该测试站点以确定此延迟是否显著。

【范例 16】

使用 VS 2012 创建一个使用默认示例的 ASP.NET Web 应用程序，假设该应用程序名称为 WebApplication3。使用“发布网站”工具发布该应用程序的步骤如下。

（1）在 VS 2012 中打开 WebApplication3 所在解决方案。

（2）从【解决方案资源管理器】窗格中右击 WebApplication3，选择【发布网站】命令，打开【发布 Web】对话框。

（3）新建一个名为 default 的配置文件，如图 17-3 所示。



图 17-3 新建配置文件

（4）单击【确定】按钮，再单击【下一步】按钮，从【发布方法】下拉列表中选择一种发布方式。默认提供了 5 种发布方式，分别是：Web Deploy、Web Deploy 包、FTP、文件系统和 FPSE。在这里选择【文件系统】选项，将网站发布到本地磁盘。

（5）单击【目标位置】文本框后的按钮，在弹出的【目标位置】对话框中指定一个网站发布后存放的目录，这里为 C:\inetpub\wwwroot\WebSite1，如图 17-4 所示。

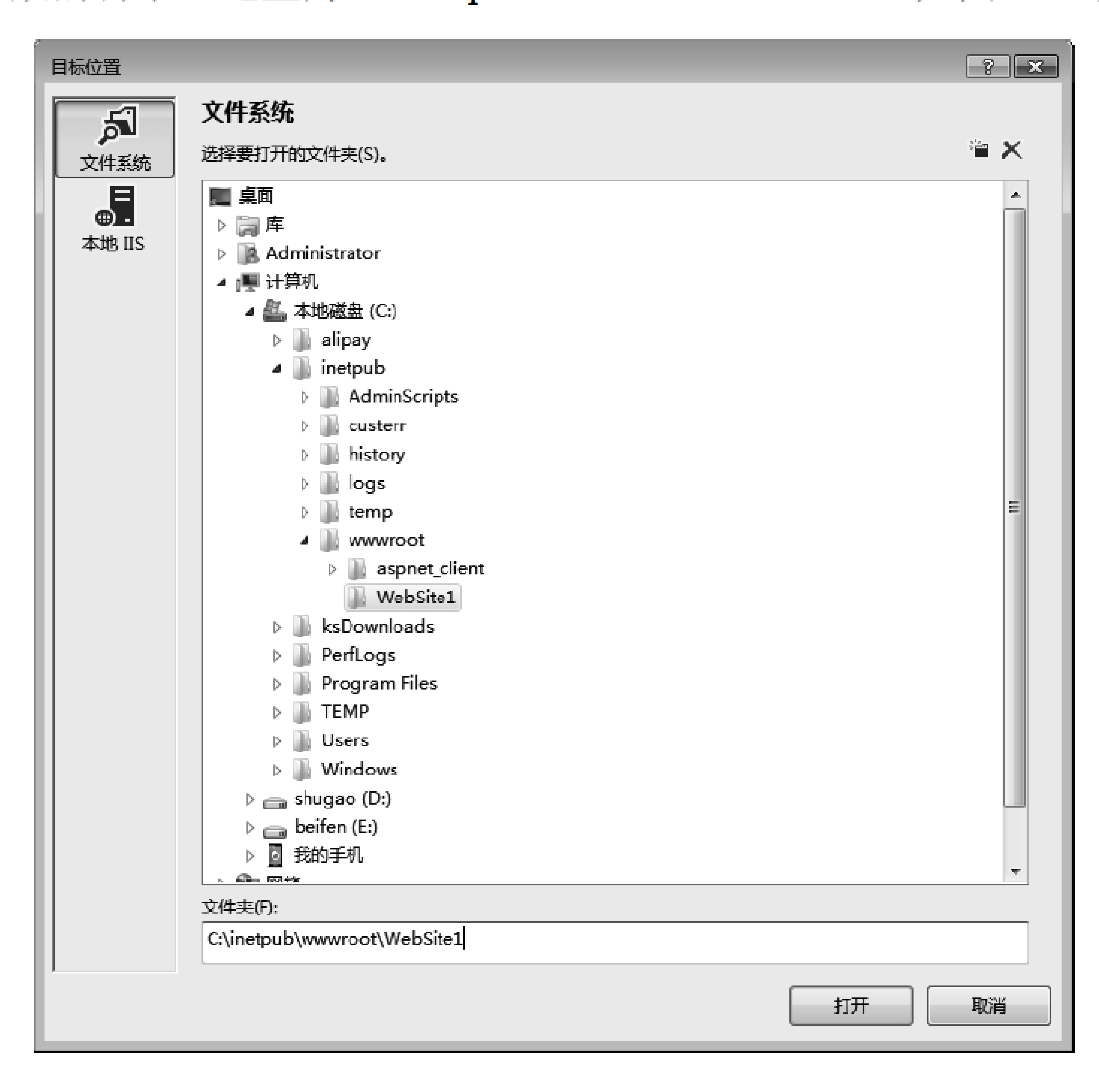


图 17-4 指定网站存放目录

（6）然后为指定的目录设置一个目标 URL，该 URL 必须能访问到网站目录，否则将出错。这里设置目标 URL 为 http://localhost，如图 17-5 所示。

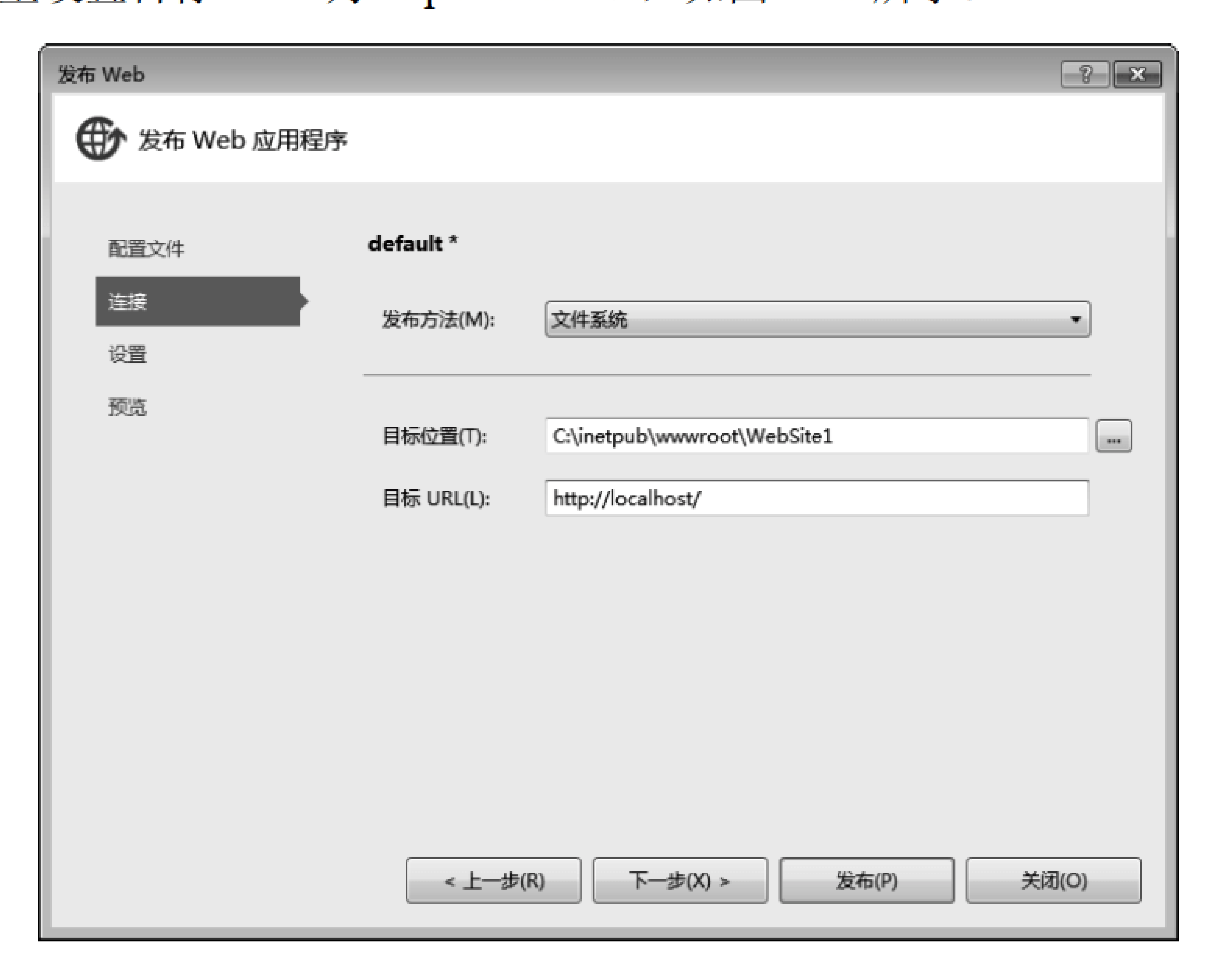


图 17-5　设置目标 URL

（7）单击【下一步】按钮，从【配置】列表中选择 Release 选项，即配置为发布版本，该版本通常对代码和配置进行了优化，使得应用程序可以更快地运行。另一个选项是 Debug，即配置为调试版本，该版本通常用于开发时，可以帮助开发人员跟踪和调试程序，并不做任何优化，如图 17-6 所示。



图 17-6　指定配置信息

（8）单击【下一步】按钮，进入发布前的预览界面，如图 17-7 所示。

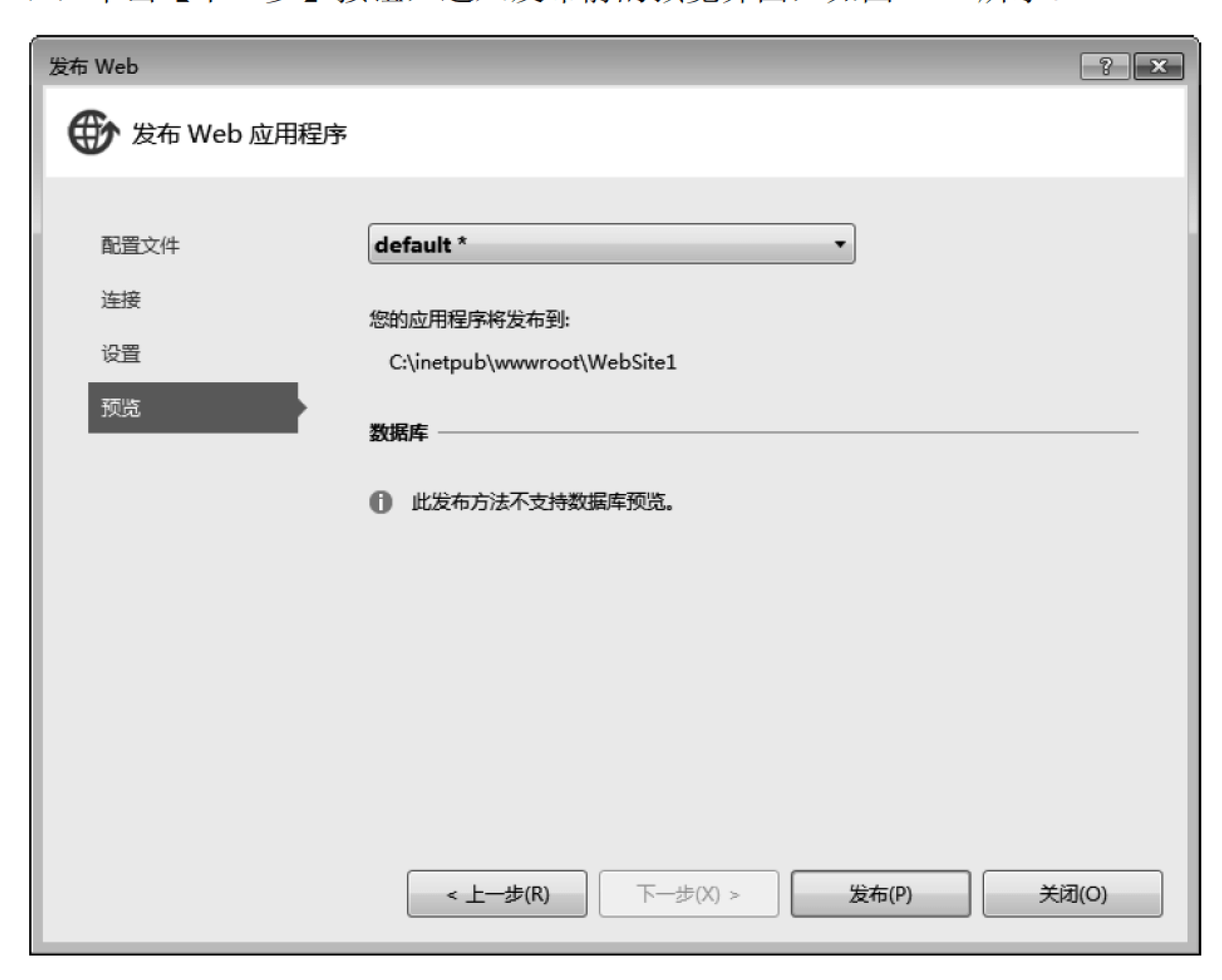


图 17-7　预览发布

（9）单击【发布】按钮确认发布。在发布过程中 VS 2012 的【输出】窗格中会实时显示正在发布的文件夹，以及最终的发布状态，如图 17-8 所示。

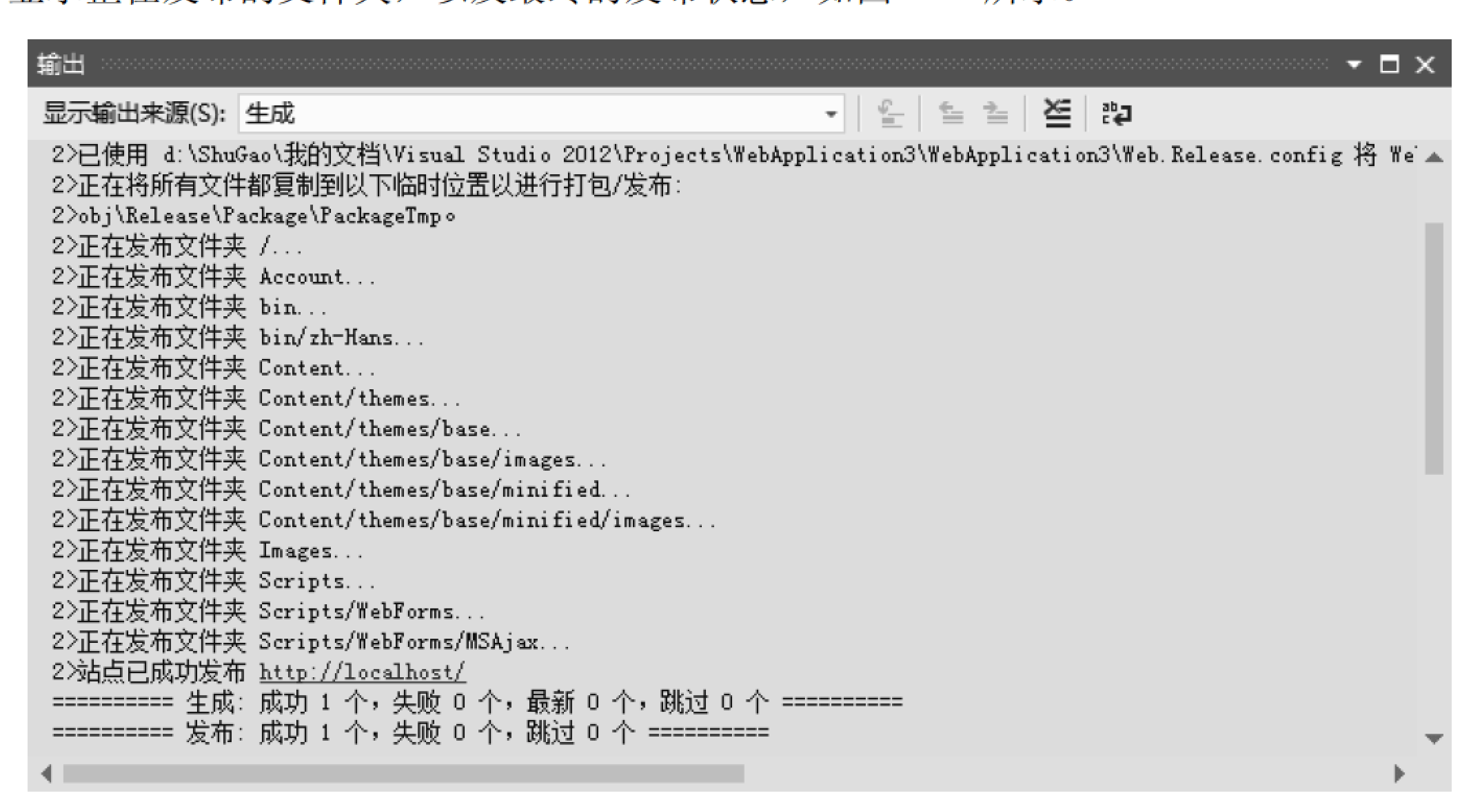


图 17-8　查看发布内容

（10）发布成功之后，打开 C:\inetpub\wwwroot\WebSite1 目录，将看到发布之后输出的最终文件。

【范例 17】

下面在 IIS 下对发布后的网站进行测试，并设置为发布时指定的 URL。这里以 Windows 7 系统下的 IIS 为例，具体安装过程忽略。步骤如下。

（1）从控制面板中打开 Internet 信息服务管理器（简称 IIS）。

（2）从左侧窗格中展开【网站】节点下的 Default Web Site 选项，从右侧窗格中单击【基本设置】链接。

（3）在弹出的【编辑网站】对话框中将网站物理路径设置为 C:\inetpub\wwwroot\WebSite1，如图 17-9 所示。



图 17–9 设置默认网站的根目录

（4）指定路径之后单击【确定】按钮。由于使用 VS 2012 发布的网站是使用的.NET Framework 4.0 版本，而 IIS 默认使用的是.NET Framework 2.0，所以此时浏览网站将会看到错误。解决方法是，在 IIS 的左侧窗格选中最顶层的节点（默认为当前机器名称），再从右侧窗格中单击【更改.NET Framework 版本】链接，在弹出的对话框中选择 4.0 版本，如图 17-10 所示。

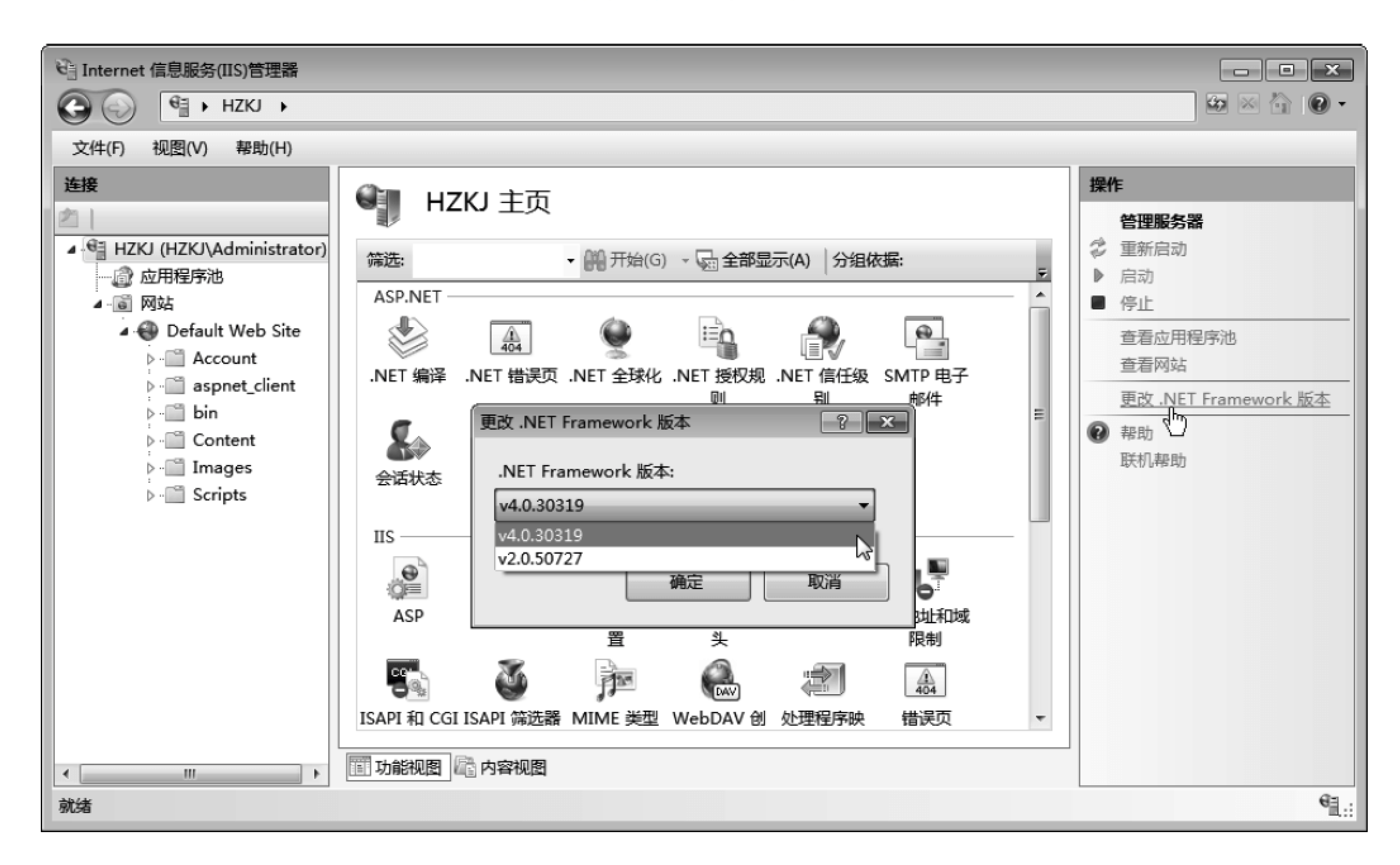


图 17-10 更改 IIS 的默认.NET Framework 版本

（5）更改默认网站使用应用程序池的.NET Framework 版本。方法是从左侧窗格中选择【应用程序池】节点，在【功能视图】列表中选择 DefaultAppPool 选项，再从右侧窗格中单击【设置应用程序池默认设置】链接。

（6）在打开的【应用程序池默认设置】对话框中，将.NET Framework 版本更改为 V4.0，如图 17-11 所示。



图 17-11 更改应用程序池的.NET Framework 版本

【范例 18】

除了上面的几个步骤外，还需要在本机上为 IIS 安装.NET Framework 4.0。方法是打开【命令提示符】窗口，进入.NET Framework 4.0 的安装目录。以 Windows 7 系统为例，目录为 C:\Windows\Microsoft.NET\Framework\v4.0.30319，命令如下。

```
cd C:\Windows\Microsoft.NET\Framework\v4.0.30319
```

再运行如下命令进行安装。

```
aspnet_regiis.exe -i
```

安装成功之后，将看到如图 17-12 所示提示信息。

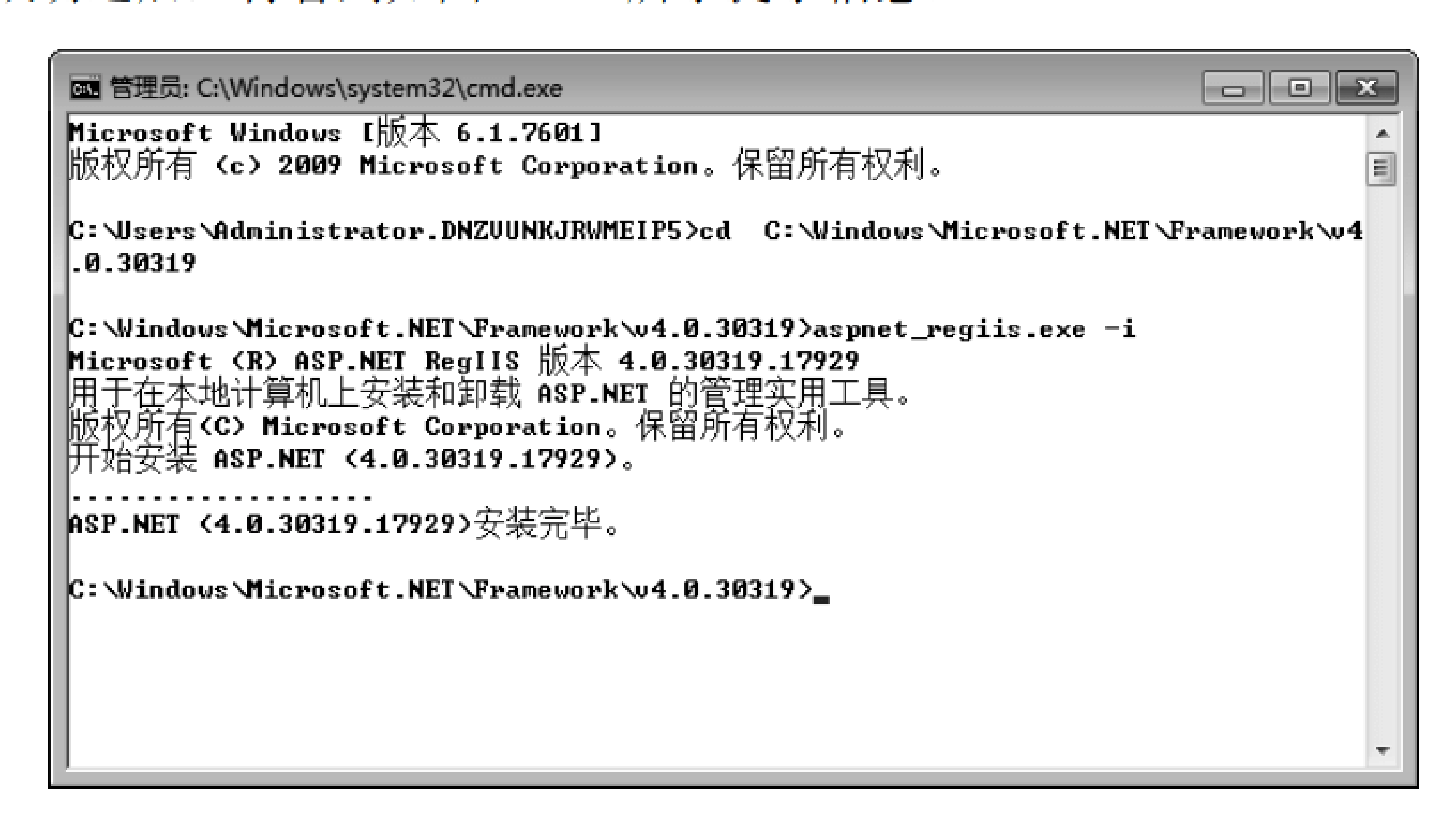


图 17-12　安装成功

在浏览器中输入 http://localhost/default.aspx，将看到发布后网站的运行效果，如图 17-13 所示。



图 17-13　发布后网站运行效果

17.3.2 通过“复制网站”工具发布

“复制网站”工具可以帮助自动完成在打开的网站项目和其他站点之间复制和同步文件的过程。利用复制网站工具，可以打开目标站点上的文件夹（它可能在远程计算机上或仅是同一计算机上的不同文件夹），然后在源网站和目标网站之间复制文件。

“复制网站”工具主要支持以下功能。

（1）可以将源文件（包括.aspx 文件和类文件）复制到目标站点，在目标服务器上请求网页时动态编译这些网页。

（2）可以使用 Visual Studio 所支持的任何连接协议复制文件。这包括本地 IIS、远程 IIS 和 FTP。如果使用 HTTP，则目标服务器必须具有 FrontPage 服务器扩展。

（3）同步功能检查源网站和目标网站中的文件，通知每个站点中哪些文件较新，并使开发人员能够选择要复制哪些文件以及要按照哪个方向复制它们。

（4）在复制应用程序文件之前，此工具将名为 App_offline.htm 的文件放置在目标网站的根目录中。当 App_offline.htm 文件存在时，对网站的任何请求都将重定向到该文件。该文件显示一条友好消息，让客户端知道正在更新网站。复制完所有网站文件后，这种工具从目标网站删除 App_offline.htm 文件。

通过“复制网站”工具发布网站项目时有两种方式：一种是选择当前网站后右击选择【复制网站】菜单项；另一种是在菜单栏中选择【网站】|【复制网站】菜单项进行发布。

【范例 19】

通过“复制网站”的方式将一个 ASP.NET 网站发布到指定的磁盘目录下。操作步骤如下。

（1）打开要发布的 ASP.NET 网站。从菜单栏中选择【网站】|【复制网站】菜单项，如图 17-14 所示。

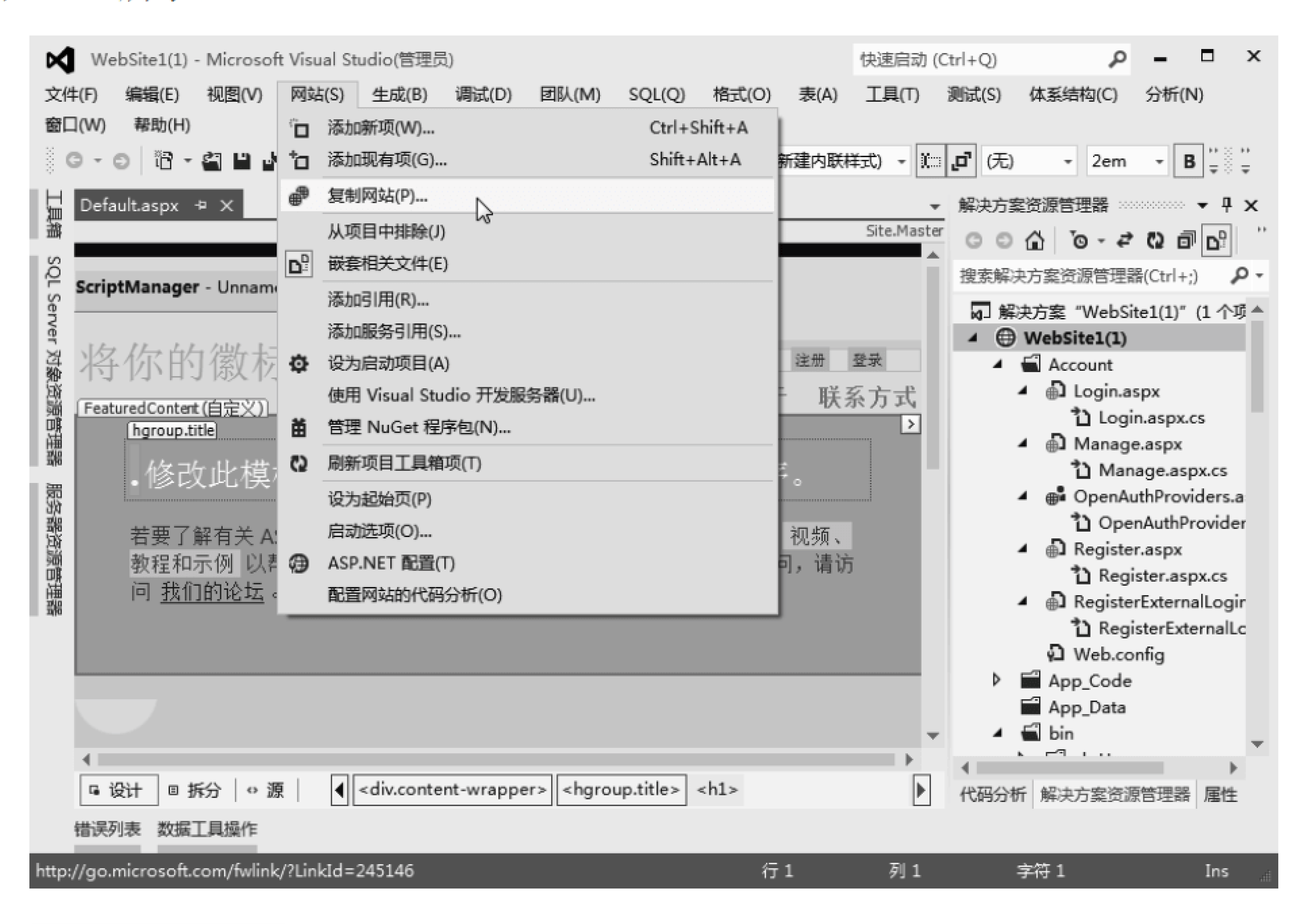


图 17-14 在菜单栏中找到【复制网站】菜单项

（2）这时会打开【复制网站】窗格，单击【连接】按钮，在弹出的【发布网站】对话框指定路径为 C:\inetpub\wwwroot\WebSite2。

（3）从【源网站】列表中选中所有文件，然后再单击相关的按钮复制到右侧的远程网站，复制完成后的效果如图 17-15 所示。

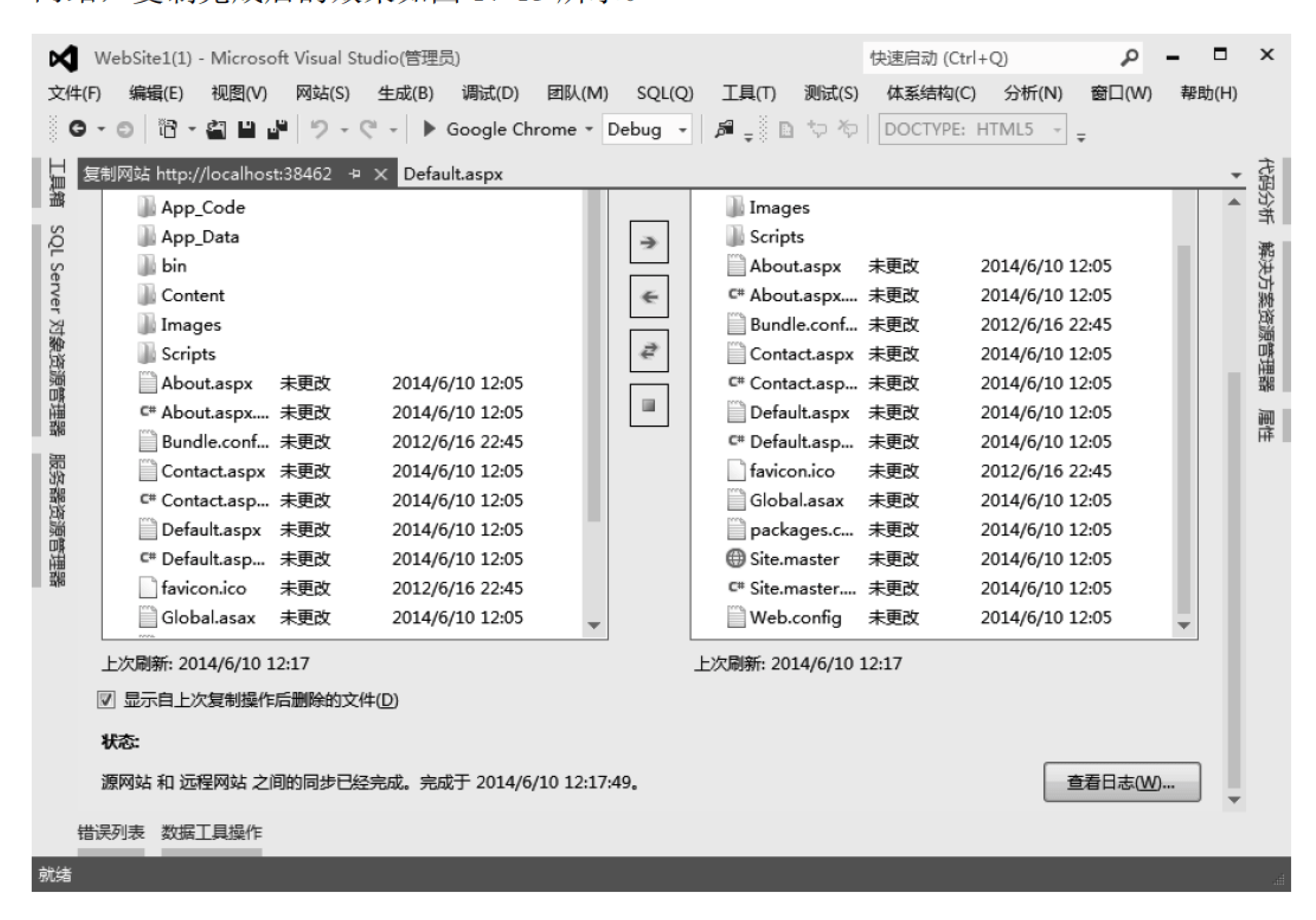


图 17-15　复制网站完成后的效果

（4）发布完毕后，在 IIS 服务器中找到 C:\inetpub\wwwroot\WebSite2 目录，打开该目录下的所有文件并且浏览查看，具体的效果图不再显示。

17.4 实验指导——通过 XCOPY 工具进行发布

除了发布网站和复制网站外，还可以通过 XCOPY 工具来发布网站。与前两种方式相比，XCOPY 是最简单的一种部署 Web 应用程序的方法。事实上，ASP.NET 的部署本身就是将页面文件、资源文件和程序集等内容复制到站点目录下，这一点与“复制网站”工具一致。

使用 XCOPY 复制网站时，有以下两种语法形式。

```
xcopy /I /s 源目标 目标目录
xcopy 源目录 目标目录 /f /e /k /h
```

无论是源目录还是目标目录，对于 XCOPY 工具来说，必须使用物理目录名称而不是虚拟目录名。

在使用 COPY 工具之前首先需要打开命令提示符窗口，然后输入命令进行复制操作。

```
XCOPY C:\inetpub\wwwroot\WebSite2 E:\WebSite2 /f /e /k /h
```

执行上述命令之后，会提示指定的目标是文件还是目录，这里输入“D”，复制过程如图 17-16 所示。复制成功后的效果如图 17-17 所示。

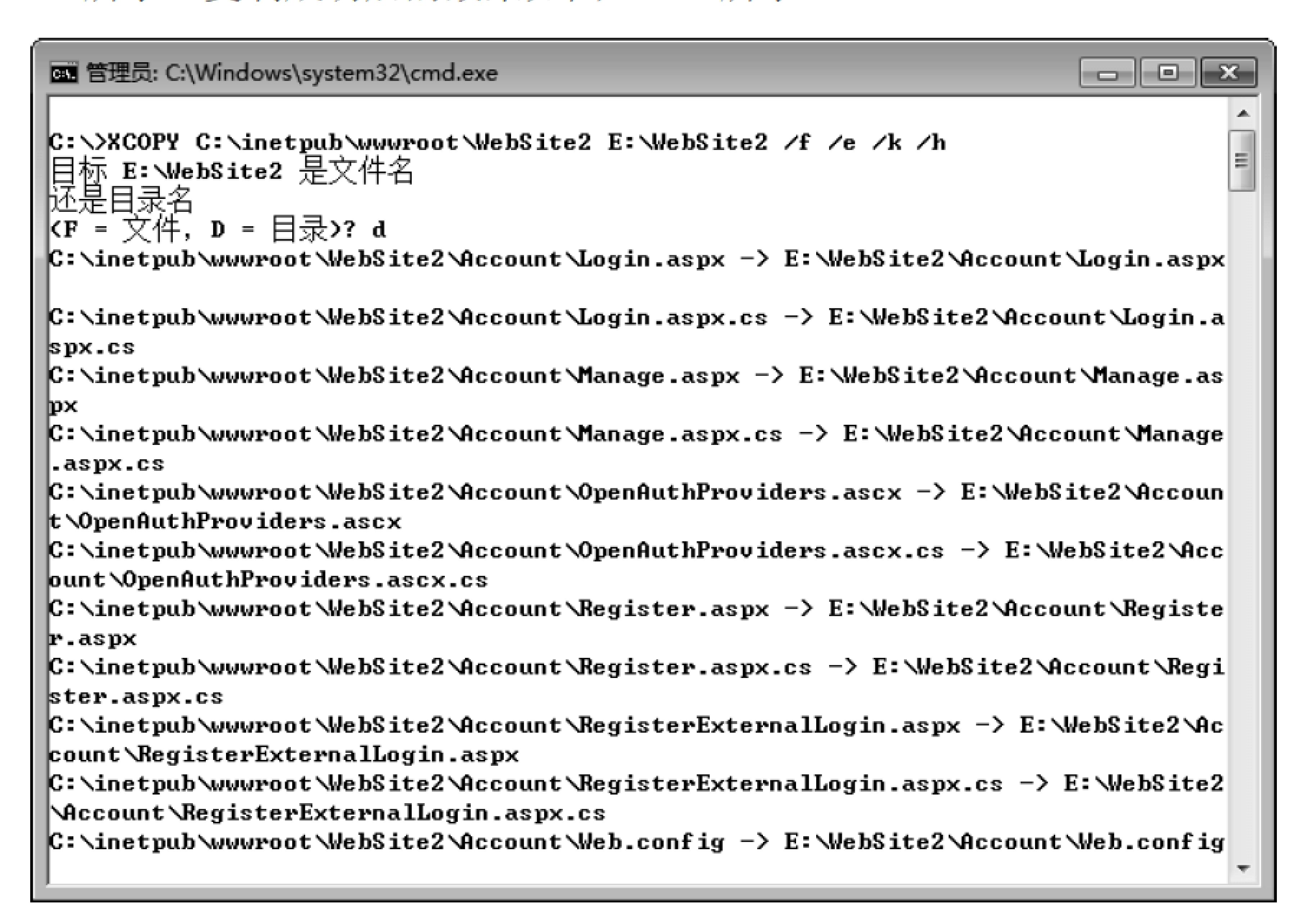


图 17-16 进行复制操作

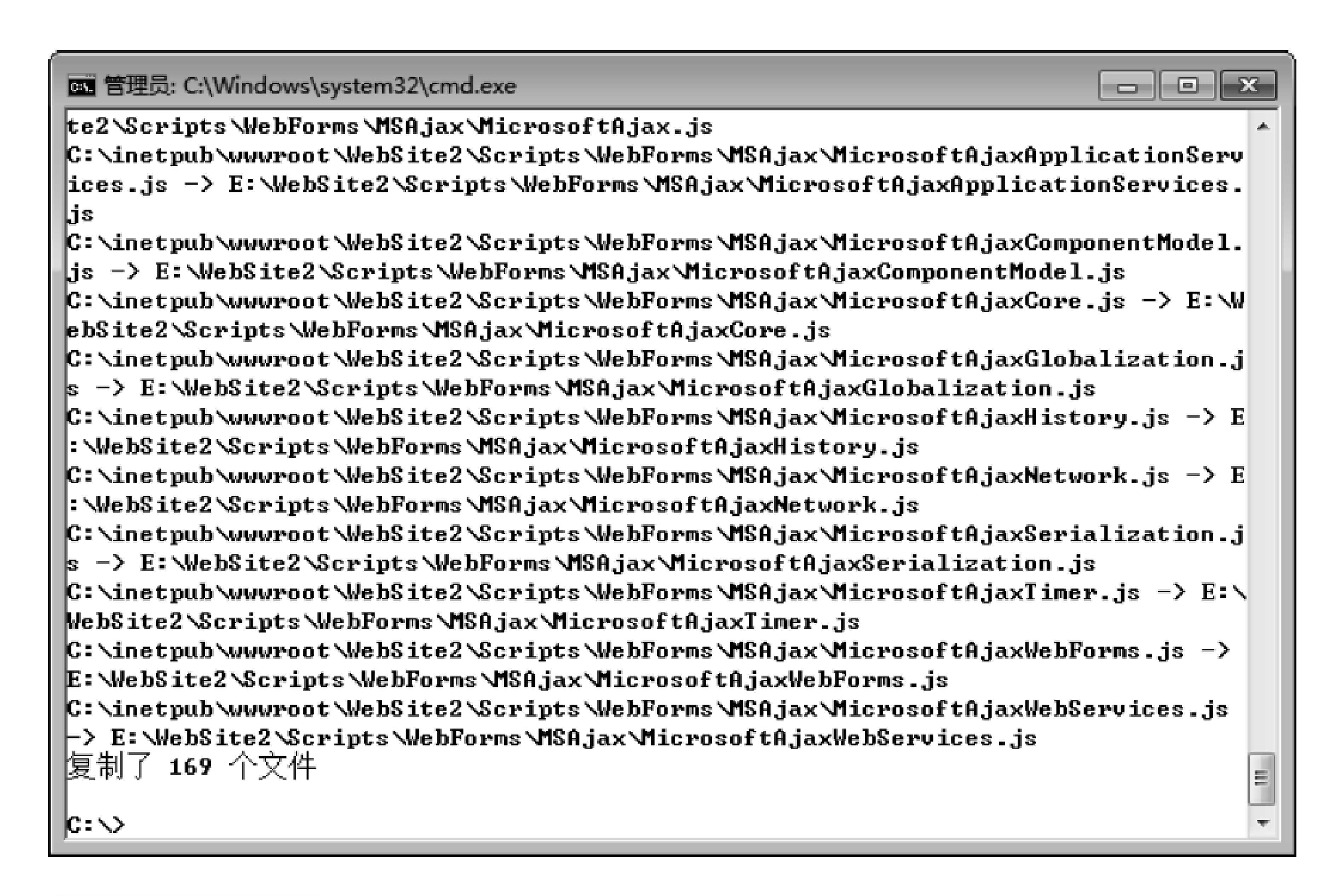


图 17-17 复制操作成功

通常情况下，通过 XPOCY 工具发布网站成功后，如果发现原来的项目有一个地方

出错需要进行更改，这时不必再将所有的文件重新复制一次，而是直接将部署或更新 Web 应用程序中的单个文件即可。

在打开的 DOS 窗口中执行以下形式的命令。

```
xcopy 源路径 目标路径
```

例如，执行以下命令将一个驱动器上\bin 目录中的一个 DLL 复制到另一个驱动器上的\bin 目录下。

```
xcopy c:\inetpub\wwwroot\devapp\bin\sampleAssembly.dll d:\publicsites\
liveapp\bin
```

执行 xcopy 命令也支持通配符，执行命令，将一个驱动器上\bin 目录下的所有 DLL 文件复制到另一个驱动器上的\bin 目录下。代码如下。

```
Xcopy c:\inetpub\wwwroot\devapp\bin\*.dll d:\publicsites\liveapp\bin
```

思考与练习

一、填空题

1. ＿＿＿＿＿文件包含整个服务器的配置信息。

2. 身份验证的类型包括 Windows、＿＿＿＿＿、Passport 和 None。

3. <sessionState>配置节中＿＿＿＿＿属性指定在用户无操作时超时的时间。

二、选择题

1. 下面选项＿＿＿＿＿不是 Web.config 文件的优点。

A. Web.config 文件基于 XML 文件类型，所有的配置信息都存放在 XML 文本文件中，可以使用文本编辑器或者 XML 编辑器直接修改和设置相应配置节

B. Web.config 配置文件具有很强的扩展性，通过该文件，开发人员能够自定义配置节

C. 开发人员可以对 Web.config 文件进行加密操作而不会影响到配置文件中的配置信息，因此保密性比较好

D. 由于 Web.config 配置文件通常存储的是 ASP.NET 应用程序的配置，所以 Web.config 配置文件的安全性较低

2. 在子目录 Admin 文件夹的 Web.config 配置文件中包括如下代码，下面这段代码说明了＿＿＿＿＿。

```
<authorization>
    <deny users="*"/>
</authorization>
```

A. 只有管理员可以访问 admin 目录

B. 所有匿名用户都可以访问 admin 目录

C. 所有匿名用户都不可以访问 admin 目录

D. 所有用户都不可以访问 admin 目录

3. ＿＿＿＿＿配置节不仅能够指定当出现错误时系统自动跳转到一个错误发生的页面，同时也能够为应用程序配置是否支持自定义错误。

A. <customErrors>

B. <sessionState>

C. <configuration>

D. <appSettings>

三、简答题

1. 请说出 Web.config 配置文件与 Machine.config 文件的区别。

2. ASP.NET 的身份验证类型有哪些？请进行解释。

3. 授权中与<allow>和<deny>配置节有关的主要属性。

4. 分别描述网站部署的三种方式。

5. 说出 Web 站点管理工具的简单使用。

附录　思考与练习答案

第 1 章　搭建 ASP.NET 4.5 的开发环境

一、填空题

1．公共语言运行时
2．System
3．Visual Studio

二、选择题

1．C
2．A
3．B
4．D

三、简答题

略

第 2 章　ASP.NET Web 窗体页

一、填空题

1．代码隐藏页
2．事件处理
3．@Page
4．TypeName

二、选择题

1．D
2．C
3．A
4．A
5．B

三、简答题

略

第 3 章　Web 服务器控件

一、填空题

1．CommandName
2．Checked
3．AlternateText
4．Multiple

二、选择题

1．C
2．A
3．B
4．D

三、简答题

略

第 4 章　页面请求与响应对象

一、填空题

1．IsPostBack
2．Redirect()
3．Platform
4．FilePath
5．MachineName

二、选择题

1．C
2．C
3．A
4．B
5．D
6．B

7. D

三、简答题

略

第 5 章　数据保存对象

一、填空题

1. Application
2. Timeout
3. Name
4. Cookies
5. ViewState

二、选择题

1. A
2. C
3. B
4. C

三、简答题

略

第 6 章　站点导航控件

一、填空题

1. SiteMapPath 控件
2. Web.sitemap
3. siteMap
4. XML 文件

二、选择题

1. D
2. D
3. A
4. C

三、简答题

略

第 7 章　使用母版页

一、填空题

1. master
2. @ Master
3. ContentPlaceHolder
4. aspx
5. .skin
6. <system.web>

二、选择题

1. D
2. B
3. A
4. C
5. C

三、简答题

略

第 8 章　验证用户输入的有效性

一、填空题

1. RequiredFieldValidator
2. Dymatic
3. MaximumValue
4. ValidationExpression
5. ServerValidate
6. ValidationGroup

二、选择题

1. A
2. C
3. B
4. B

5．D
6．D

三、简答题

略

第 9 章　ADO.NET 进行数据库编程

一、填空题

1．DataSet
2．CreateCommand()
3．ExecuteNonQuery()
4．ParameterName
5．Read()
6．Fill()

二、选择题

1．A
2．C
3．D
4．D
5．A
6．B

三、简答题

略

第 10 章　数据绑定技术

一、填空题

1．Eval()
2．SiteMapDataSource
3．RepeatColumns
4．AllowSorting

二、选择题

1．D
2．A
3．B
4．C
5．B

三、简答题

略

第 11 章　LINQ 数据处理

一、填空题

1．System.Linq
2．Querable
3．from
4．DatabaseExists()
5．Connection

二、选择题

1．A
2．C
3．D
4．C
5．A

三、简答题

略

第 12 章　高级技术应用

一、填空题

1．FileUpload
2．File
3．Extension
4．Exists

二、选择题

1．C

2．B
3．C
4．C
5．D

三、简答题

略

第 13 章　Ajax 技术

一、填空题

1．XmlHttpRequest
2．Timer
3．UpdateProgress
4．slideDown()
5．单击
6．XPath

二、选择题

1．D
2．C
3．A
4．C
5．D
6．B
7．D

三、简答题

略

第 14 章　Silverlight 入门

一、填空题

1．XAP
2．System.Windows.Windows.UserControl
3．xmlns
4．System.Windows.Browser
5．HtmlPage.Document
6．System.Windows.Threading

二、选择题

1．B
2．C
3．A
4．A
5．A
6．D

三、简答题

略

第 15 章　ASP.NET MVC 4 框架

一、填空题

1．控制器
2．Controllers
3．Global.asax
4．@{
5．@Html.LabelFor(m => m.UserName)
6．RouteConfig.cs

二、选择题

1．C
2．A
3．A
4．B
5．A
6．A

三、简答题

略

第 16 章　WCF 入门

一、填空题

1．.NET Framework 3.0
2．Host
3．地址
4．ServiceContract
5．IHttpHandler

二、选择题

1．A
2．D
3．B
4．C
5．D

三、简答题

略

第 17 章　配置和部署 ASP.NET 网站

一、填空题

1．Machines.config
2．Forms
3．timeout

二、选择题

1．D
2．C
3．A

三、简答题

略